Honest Majority and Beyond: Efficient Secure Computation over Small Population

A THESIS

SUBMITTED FOR THE DEGREE OF

Master of Technology (Research)

IN THE

Faculty of Engineering

BY

Swati Singla

Computer Science and Automation
Indian Institute of Science
Bangalore – 560 012 (INDIA)

June, 2019

Declaration of Originality

I, **Swati Singla**, with SR No. **04-04-00-10-22-16-1-13894** hereby declare that the material presented in the thesis titled

Honest Majority and Beyond: Efficient Secure Computation over Small Population

represents original work carried out by me in the **Department of Computer Science and Automation** at **Indian Institute of Science** during the years **2016-2019**.
With my signature, I certify that:

- I have not manipulated any of the data or results.
- I have not committed any plagiarism of intellectual property. I have clearly indicated and referenced the contributions of others.
- I have explicitly acknowledged all collaborative research and discussions.
- I have understood that any false claim will result in severe disciplinary action.
- I have understood that the work may be screened for any form of academic misconduct.

Date: Student Signature

In my capacity as supervisor of the above-mentioned work, I certify that the above statements are true to the best of my knowledge, and I have carried out due diligence to ensure the originality of the report.

Advisor Name: Advisor Signature

DEDICATED TO

My beloved

Maa & Papa

Acknowledgements

The first person I would like to extend my most sincere gratitude to is my advisor Dr. Arpita Patra for welcoming me to the 'Cryptography and Information Security' lab. Other than being my research mentor and being solely responsible for my academic progress, there are a shedload of personally invaluable lessons she has added to my life. She is a personality with unparalleled research enthusiam topped with childlike innocent curiosity. Her motherly love for the lab members and her steadfast ethics are what glued the group together. I would consider my life a tremendous success if I someday happen to become half the woman she is.

I am joyously indebted to my labmates (some of which were my co-authors for my research submissions) for the brilliant brainstorming sessions. The cheerful lab atmosphere and the knowledge transferred inside the lab as a result of active detailed discussions was massively crucial to the work in this thesis. In particular, I would like to thank Divya Ravi, my personal go-to research guru, who taught me the meticulous art of research. I feel supremely fortunate to have got the experience of working closely with her on a project that was indispensable in building my research aptitude. The next person I would like to extend my love to is Megha Byali, a partner in all my projects and one of my closest friends at IISc. She was my rock when I had hit the depressing researcher's block. She had the unbelievable talent of making work look simple and life was easy around her!

A shout-out to my beloved friends, Vertika Singh, Aishwarya Garg, Kunal Chelani and Nishant Bhai for standing strong with me in this entire journey. These people are my safe home, having seen me at my worst. Also, my most adorable pals I spent the time of my life with in terms of the countless drives and dances we embarked upon together in my first year: Sanket Purandare, Stanly Samuel, Bhushan Patil and Ajinkya Rajput.

Lastly, I would like to extend immense love to my family: a father who fulfilled my every single (even stupid) wish, a mother who was my strength, a brother whose rock-solid support took my worries away in a jiffy and the recent addition, a sister-in-law who has made this home a merrier place. I will forever be indebted to Maa, who has gone beyond her means all my life to give me extraordinary education and values and to nurture all my silly impulsive passions.

Abstract

Secure Multi-Party Computation for small population has witnessed notable practically-efficient works in the setting of both honest majority and dishonest majority. While honest majority provides the promise of stronger security goals of fairness (either all parties get the output or none of them do) and guaranteed output delivery (honest parties always get the output irrespective of adversary's behaviour), the best that dishonest majority can offer is unanimous abort (either all honest parties get the output or none of them do). In this work, we consider the computation among 4 parties in two different threat models. To avoid clutter and enable ease of understanding, we segregate the thesis into two parts (one for each threat model).

Part I considers the standard honest majority (i.e. 1 corruption) where we provide constant-round (low-latency) protocols in a minimal model of pairwise private channels. Improving over the state-of-the-art work of Byali et al. (ACM CCS '18), we present two instantiations that efficiently achieve: (a) fairness in 3 rounds using 2 garbled circuits (GC) (b) guaranteed output delivery (GOD) in 3 rounds using 4 GCs. Further, improving the efficiency of 2-round 4PC feasibility result of Ishai et al. (CRYPTO '15) that achieves GOD at the expense of 12 GCs, we achieve GOD in 2 rounds with 8 GCs, thus saving 4 GCs over that of Ishai et al. Under a mild one-time setup, the GC count can further be reduced to 6 which is half of what the prior work attains.

This widely-followed demarcation of the world of MPC into the classes of honest and dishonest majority suffers from a worrisome shortcoming: one class of protocols does not seem to withstand the threat model of the other. Specifically, an honest-majority protocol promising fairness or GOD violates the primary notion of privacy as soon as half (or more) parties are corrupted, while a dishonest-majority protocol does not promise fairness or GOD even against a single corruption, let alone a minority. The promise of the unconventional yet much sought-after brand of MPC, termed as Best-of-Both-Worlds (BoBW), is to offer the best possible security in the same protocol depending on the actual corruption scenario. With this motivation in perspective, part II presents two practically-efficient 4PC protocols in the BoBW model, that achieve: (1) guaranteed output delivery against 1 corruption and unanimous abort against 2

corruptions. (2) fairness against 1 corruption and unanimous abort against arbitrary corruptions. The thresholds are optimal considering the feasibility given in the work of Ishai et al. (CRYPTO '06) that marks the inauguration of the BoBW setting.

We provide elaborate empirical results through implementation that support the theoretical claims made in all our protocols. We emphasize that this work is the first of its kind in providing practically-efficient constructions with implementation in the BoBW model. Also, the quality of constant-rounds makes all protocols in this work suitable for high-latency networks such as the Internet.

Publications

Based on the thesis

- Megha Byali, Arpita Patra, Divya Ravi and **Swati Singla**. *Beyond Honest Majority: On the Efficiency of 4-Party Computation in High-latency Networks.* Under Submission.

- Megha Byali, Nishat Koti, Arpita Patra, Divya Ravi and **Swati Singla**. *Speedo4: High-Speed Secure 4-Party Computation over the Internet.* Under Submission.

Other

- Megha Byali, Carmit Hazay, Arpita Patra and **Swati Singla**. *Fast Actively-secure Five Party Computation with Security Beyond Abort.* **ACM CCS 2019**.

- Arpita Patra, Divya Ravi and **Swati Singla**. *On the Exact Round Complexity of Best-of-both-Worlds Multi-Party Computation.* Under Submission.

Contents

List of Figures

List of Tables

Part I

4PC in Honest-Majority Setting

Chapter 1

Introduction

Protocols for secure multiparty computation (MPC) [Yao82, GMW87, CDG87] allow a set of mutually distrusting parties to compute a function in a distributed fashion while guaranteeing the *privacy* of the parties' inputs and the *correctness* of their outputs such that no coalition of t parties can violate either correctness or privacy. The vast body of research in this area can be divided into two, almost disjoint, lines of work: one when a majority of the parties are assumed to be honest [BGW88, RB89, BMR90, DN07, BH07, BH08, BFO12], and the other when an arbitrary number of parties may be corrupted [GMW87, DO10, BDOZ11, DPSZ12, AJL+12, GGHR14, LPSY15]. Efforts have been made in both areas to achieve practical constructions. In the dishonest-majority world, 2PC has been of special interest in the past decade due to the efficiency factor. However, MPC with small population in honest majority has been the cornerstone of research lately mainly due the extensive applicability it offers for real-time systems. Most real-time MPC implementations such as financial data analysis [BTW12], danish sugar beet auction [BCD+09], distributed credential encryption [MRZ15], server aided computation [MR18], involve small number of parties. Also, MPC with small parties is a major contributor in the recent advancement of utilizing machine learning as a prediction service offered by technology giants such as Amazon Web Services. Attaining this service in a privacy preserving manner involves outsourcing the computation to servers in a distributed way, with computation performed using MPC [DSZ15, MR18]. Further, when a majority of the participants are honest, it is possible to attain the strong security promises such as *guaranteed output delivery* (GOD) (honest parties are guaranteed to successfully complete the computation) and *fairness* (either all parties get the output or none of them do) [Cle86]. However, when no honest majority is assumed, a weaker notion *security with abort* is used. Stronger security is desirable for practical applications such as voting and auctions, to prevent the adversary from repeatedly causing the computation to rerun, thereby wasting valuable time and resources on one hand and on

the other, learning multiple outcomes and utilising them for frauds (such as election rigging). The recent work of [BJPR18] has demonstrated that these stronger goals can be achieved with minimal overhead over their abort counterparts. Further, honest majority aids in obtaining simple yet efficient constructions with the use of inexpensive tools such as customized secret sharing, semi-honest Yao circuits and so on as opposed to the inevitability of expensive public-key primitives such as oblivious transfer in dishonest-majority setting. All these features make the study of MPC for small population in honest-majority worthwhile.

This work explores the efficiency of four-party (4PC) protocols tolerating one active corruption achieving strong security of *fairness* and *GOD*. The 4PC setting is quite special for reasons below: (1) Ensuring honest majority and satisfying $n = 3t + 1$ at the same time eliminates the need of broadcast channel to achieve GOD, making the constructions highly efficient. However, broadcast is inevitable in three-party (3PC) owing to the result of [CL14]. (2) For any message sent by a party that needs agreement, a simple honest majority rule by the residual three parties suffices. Such a property cannot be counted on in 3PC. These key features are leveraged in our work, proving the simplicity and efficiency gains of 4PC over 3PC.

1.1 Related Work

The notable works in MPC with small population can be broadly cast into high-throughput and low-latency protocols. The former aims at minimizing the communication overhead (bandwidth) at the expense of non-constant rounds while the latter involves low-latency constructions that are constant round and are best suited for Internet like networks. In this paper, we focus on the latter category and highlight the most relevant literature below.

In the regime of constant-round protocols, the work of [MRZ15] presents a 3PC with selective abort in 3 rounds. Concurrently, [IKKP15] also presents a 3PC with selective abort and additionally a 4PC with GOD in 2-rounds at the expense of 12 garbled circuits. The work of [PR18] presents theoretical lower bounds of honest majority protocols achieving stronger security notions and matching upper bounds for 3PC. Most recently, [BJPR18] presents efficient 3PC and 4PC with stronger security notions of fairness and GOD with minimal overhead over the state of the art [MRZ15]. For corruptions beyond 1, [CGMV17] presents an efficient selective abort construction for 5 parties. The work of [CKMZ14] explores 3PC in the dishonest majority. In the domain of high-throughput protocols, [GRW18] explores 4PC with abort, fairness and GOD. In the regime of 3PC with honest-majority, [AFL+16] presents an abort protocol that operates invariably over rings and fields tolerating passive corruption. The work of [CGH+18] proposes a compiler to convert passive security to active at twice the cost of the passive protocol. In the active setting, the works of [ABF+17, DOS18, FLNW17, BBC+19]

consider computation over circuits defined on rings. The recent work of [NV18] improves the state of the art [CGH+18] by presenting efficient techniques for batch-multiplication.

1.2 Our Contributions

In the regime of low-latency protocols, we propose efficient protocols that achieve the stronger security of *fairness* and *GOD* for 4 parties tolerating one malicious corruption. We attempt to improve the state-of-the-art [BJPR18] and [IKKP15] in terms of rounds and communication. All our garbled circuit (GC) based protocols are set in a minimal network with parties connected *only* via pairwise-private channels. We summarize our contributions below.

3 round protocols We present two 3-round protocols achieving: (1) *fairness* and (2) *GOD* at the expense of 2 GCs and 4 GCs respectively. Both protocols involve two garblers and two evaluators to leverage the presence of at least one honest evaluator. Performance wise, our 3-round protocols improve over [BJPR18] and are highly efficient in WAN. On a technical note, in our fair protocol, we adopt the idea of [MRZ15] that asks two parties to emulate the garbler role for active security. We use replicated secret sharing (RSS) for input distribution. The first 2 rounds of input distribution are overlapped with the garbling phase to optimize rounds which introduces multiple challenges. Inspired from [BJPR18], we rely on *oblivious* garbling and withhold the decoding information until it is ensured that the circuit evaluation is performed on the correct input only. However with only one round remaining for robust output computation, it is quite tricky to decide whether (or not) to release the decoding information while ensuring agreement on output among all honest parties. We address such concerns by putting together delicate techniques such as splitting the decoding information, strategic use of signals to indicate the legitimacy of the output and means for an evaluator to aid her co-evaluator to obtain encoding information without compromising on security. Building on our fair protocol, we achieve GOD by tackling the cases leading to abort, without inflating the round complexity. For each abort situation, we enable a pair of honest parties to unanimously conclude by the end of round 2 about either (a) the corrupt party or (b) two parties in conflict. To ensure the identified corrupt (or conflicting) party obtains the output, the honest pair generates *one* extra GC for case (a) and *two* for case (b) and send GCs along with the encoding and decoding information to the corrupt (or conflicting) party. Additionally, the pair exchanges their views to construct the output amongst themselves (2 parties suffice to compute any input as per RSS).

2 round protocols We improve the feasibility result of [IKKP15] that achieves GOD in 2 rounds at the expense of 12 GCs and present a construction that achieves GOD at the expense

of 8 GCs while preserving the round optimality. We first present a construction that achieves GOD in 2 rounds with 6 GCs assuming the presence of a mild one-time setup and then provide techniques to eliminate the setup and handle the challenges that come up. On a technical note, it can be observed that the latest a party can obtain the output is as an evaluator at the end of 2 rounds in the GC based approach. Accordingly, we run four robust 2-round executions with each party P_i obtaining output from i^{th} execution where she acts as evaluator and the remaining 3 parties as garblers. While we adopt some techniques of input distribution using RSS and circuit augmentation (to include input commitment logic) from [IKKP15], the main ideas in improving efficiency come from: (1) Use of only 4 executions with each party as evaluator exactly once as opposed to [IKKP15] that involves computation of a GC for every possible 3-party committee (12 in total; ensuring each party acts as evaluator in one all-honest committee). (2) Each execution comprises of 3 garblers and two different GCs are constructed with randomness for each GC distributed by a different garbler. This ensures the presence of at least one honest distributor and the corresponding GC can be used for robust evaluation. However, a distinctive challenge we face in our construction is that a corrupt distributor can misbehave such that, within an execution, the encoding information of an honest garbler is rendered useless. This is resolved with careful discretion by asking each garbler to additionally release the encoding information corresponding to her co-garblers' wires. This solution demands a highly-involved analysis to ensure that the input privacy and consistency is preserved across executions. Performance wise, the gain of 4 GCs over [IKKP15] turns out to be significant as depicted in Chapter 7. In essence, our protocol strikes a good balance between rounds and efficiency.

Empirical Results and Comparison The table below summarizes the overhead incurred by each of our protocols to achieve the stronger security goals: *fairness* and *GOD* when compared to state-of-the-art in terms of overall LAN runtime, WAN runtime; communication involved. The bandwidth of WAN is limited to 8Mbps as our protocols cater to systems with limited bandwidth support. The values are given for benchmark circuits of AES-128 and SHA-256 with range determined over the benchmark circuits. (**- bold numbers** indicate the gain in efficiency).

Protocol	Parameters	3PC Abort [MRZ15] 3 Rounds	3PC fair [BJPR18] 4 Rounds	3PC GOD [BJPR18] 5 Rounds	4PC GOD [BJPR18] 5 Rounds
3RFair (4PC Fair)	LAN(ms)	1.55 – 22.86	1.7 – 22.62	0.95 – 20.76	0.44 – 32.92
	WAN(s)	1.14 – 1.29	0.18 – 0.66	–	(**-0.48**) – 0.04
	Comm(KB)	0.34 – 6.08	0.34 – 6.05	0.34 – 6.08	0.32 – 6.04
3RGod (4PC GOD)	LAN(ms)	2.07 – 23.18	2.22 – 22.94	1.47 – 21.08	0.96 – 33.24
	WAN(s)	1.18 – 1.25	0.22 – 0.62	–	(**-0.44**) – 0
	Comm(KB)	0.34 – 6.08	0.34 – 6.05	0.34 – 6.08	0.32 – 6.04
3RGod (4PC GOD worst case)	LAN(ms)	6.35 – 23.62	6.5 – 23.38	5.75 – 21.52	5.24 – 33.68
	WAN(s)	1.45 – 1.58	0.62 – 0.82	–	(**-0.04**) – 0.2
	Comm(KB)	0.62 – 12.12	0.62 – 12.09	0.62 – 12.12	0.6 – 12.08

In summary, the overhead for our 3-round protocols is a consequence of the use of more than one GC. This overhead, however, is annulled by the efficiency gain in WAN resulting from minimizing the round complexity, thus bridging the gap between efficiency and optimal round complexity, which is of foremost priority in networks such as the Internet.

1.3 Outline of Part I

Post the introduction, this part of the thesis starts with the preliminaries needed for the understanding of the work. In Chapter 3, we discuss some building blocks common to all the constructions. We present the details of each protocol from the subsequent chapter: Chapter 4 presents the fair protocol in 3 rounds using 2 GCs. Chapter 5 presents the GOD protocol in 3 rounds using 4 GCs. Chapter 6 starts with the GOD construction with a mild setup and later, provides a mechanism to get rid of the setup, both the constructions requiring 2 rounds and 8 GCs. An optimisation is given to reduce the number of GCs required to 6 GCs for the protocol with setup. Each construction is backed with a security proof presented via the existence of a simulator. The final chapter provides elaborate implementation results for both LAN and WAN setting. We compare our work with the relevant constructions in the literature to provide a concrete evidence of the claimed improvement.

Chapter 2

Preliminaries

2.1 Security Model

We consider 4 parties which are connected via pair-wise secure and authentic channels and modelled as non-uniform PPT interactive Turing machines. We denote by $\mathbf{P}$ the set of 4 parties i.e. $\mathbf{P} = \{P_1, P_2, P_3, P_4\}$. We consider static security model with honest majority where a PPT adversary $\mathcal{A}$ can maliciously corrupt at most 1 party at the onset of protocol. Let κ denote the computational security parameter. The security of our protocols is proved in the real/ideal world paradigm, i.e. security of a protocol is analyzed by comparing the adversary's behaviour in the real world execution and the ideal world execution (which is considered secure by definition in the presence of an incorruptible trusted third party). In an ideal execution, the parties send their inputs to the trusted third party via a perfectly secure channel, the trusted party computes the function output on the inputs provided, and sends the respective outputs to the parties. Informally, we say that a protocol is secure if whatever an adversary can do in the real world execution can be simulated in the ideal world execution. Please refer to [Can00, Gol01, CL14, Lin17] for further details.

The real world execution of a protocol Π consists of a set of parties (in our case 4) and a PPT adversary $\mathcal{A}$ which may corrupt atmost one party. The ideal world execution consists of a set of parties, an ideal world adversary $\mathcal{S}$ and a functionality $\mathcal{F}$. Let $\text{IDEAL}(1^\kappa, z)$ denote the joint output of the honest parties and $\mathcal{S}$ from the ideal execution with respect to the security parameter κ and auxiliary input z. Similarly, let $\text{REAL}(1^\kappa, z)$ denote the joint output of the honest parties and $\mathcal{A}$ from the real world execution. We say that the protocol Π securely realizes $\mathcal{F}$ if for every PPT adversary $\mathcal{A}$ there exists an ideal world adversary $\mathcal{S}$ corrupting the same parties such that $\text{IDEAL}_{\mathcal{F},\mathcal{S}}(1^\kappa, z)$ and $\text{REAL}_{\Pi,\mathcal{A}}(1^\kappa, z)$ are computationally indistinguishable.

2.2 Functionalities

We define the ideal functionalities for the security notion of GOD ($\mathcal{F}_{\mathsf{god}}$), fair ($\mathcal{F}_{\mathsf{fair}}$) and unanimous abort ($\mathcal{F}_{\mathsf{uAbort}}$) for secure 4PC of a function f in figures 2.1, 2.2 and 2.3 respectively. These are motivated from [CL14, GLS15].

Each honest P_i ($i \in [4]$) sends its input x_i to the functionality. Corrupt parties may send arbitrary inputs chosen by the adversary.

Input: On message (Input, x_i) from P_i, do the following: if $(\mathsf{Input}, *)$ already received from P_i, then ignore the current message. Otherwise, record $x_i' = x_i$ internally. If x_i is outside P_i's domain, consider x_i' to be some predetermined default value.

Output: Compute $y = f(x_1', x_2', x_3', x_4')$. Send (Output, y) to all.

Figure 2.1: Ideal Functionality $\mathcal{F}_{\mathsf{god}}$

Every honest party P_i ($i \in [4]$) sends its input x_i to the functionality. Corrupted parties may send arbitrary inputs as instructed by the adversary. When sending inputs, the adversary is allowed to send a special `abort` command.

Input: On message (Input, x_i) from P_i, do the following: if $(\mathsf{Input}, *)$ already received from P_i, then ignore the current message. Otherwise, record $x_i' = x_i$ internally. If x_i is outside P_i's domain, consider $x_i' = \texttt{abort}$.

Output: If there exists $i \in [4]$ such that $x_i' = \texttt{abort}$, send $(\mathsf{Output}, \perp)$ to all the parties. Otherwise, send (Output, y) to all parties, where $y = f(x_1', x_2', x_3', x_4')$.

Figure 2.2: Ideal Functionality $\mathcal{F}_{\mathsf{fair}}$

Each honest party P_i ($i \in [4]$) sends its input x_i to the functionality. Corrupted parties may send arbitrary inputs as instructed by the adversary. When sending the inputs to the trusted party, the adversary is allowed to send a special `abort` command as well.

Input: On message (Input, x_i) from P_i, do the following: if $(\mathsf{Input}, *)$ message was received from P_i, then ignore. Otherwise record $x_i' = x_i$ internally. If x_i' is outside of the domain for P_i, consider $x_i' = \texttt{abort}$.

Output to the adversary: If there exists $i \in [4]$ such that $x_i' = \texttt{abort}$, send $(\texttt{Output}, \perp)$ to all the parties. Else, send $(\texttt{Output}, y)$ to the adversary, where $y = f(x_1', x_2', x_3', x_4')$.

Output to honest parties: Receive either continue or abort from the adversary. In case of continue, send y to all honest parties. In case of abort send $\perp$ to all honest parties.

Figure 2.3: Ideal Functionality $\mathcal{F}_{\mathsf{uAbort}}$

2.3 Primitives

A function $\mathsf{negl}(\cdot)$ is said to be *negligible* if for every positive polynomial $\mathsf{poly}(\cdot)$ there exists a positive integer n_0 s.t $\forall n > n_0$, $\mathsf{negl}(n) < \frac{1}{\mathsf{poly}(n)}$. A *probability ensemble* $X = \{X(a,n)\}_{a\in\{0,1\}^*;n\in\mathbb{N}}$ is an infinite sequence of random variables indexed by $a \in \{0,1\}^*$ and $n \in \mathbb{N}$. Probability ensembles $X = \{X(a,n)\}_{a\in\{0,1\}^*;n\in\mathbb{N}}$, $Y = \{Y(a,n)\}_{a\in\{0,1\}^*;n\in\mathbb{N}}$ are said to be *computationally indistinguishable*, denoted by $X \overset{c}{\approx} Y$, if for every PPT algorithm $\mathcal{D}$, there exists a negligible function $\mathsf{negl}(\cdot)$ s.t for every $a \in \{0,1\}^*$, $n \in \mathbb{N}$, $|\mathsf{Pr}[\mathcal{D}(X(a,n)) = 1] - \mathsf{Pr}[\mathcal{D}(Y(a,n)) = 1]| \leq \mathsf{negl}(n)$. Next, we describe the primitives used in our constructions.

2.3.1 Collision-Resistant Hash [RS04]

A family of hash functions $\{\mathsf{H} : \mathcal{K} \times \mathcal{M} \to \mathcal{Y}\}$ is said to be collision resistant if for all PPT adversaries $\mathcal{A}$, given the hash function H_k for $k \in_R \mathcal{K}$, the following holds: $\mathsf{Pr}[(x,x') \leftarrow \mathcal{A}(k) : (x \neq x') \wedge \mathsf{H}_k(x) = \mathsf{H}_k(x')] = \mathsf{negl}(\kappa)$, where $x, x' \in \{0,1\}^m$ and $m = \mathsf{poly}(\kappa)$.

2.3.2 Replicated Secret Sharing (RSS) [CDI05b, ISN89]

We use a 3-party RSS scheme private against one corruption. RSS allows a dealer to share a secret among a set of parties s.t any two shareholders can come together and reconstruct the secret, but a single party, by itself, will have no information about the secret s. Informally, for a secret s to be shared, the dealer samples random r_1, r_2, r_3 s.t. $s = r_1 \oplus r_2 \oplus r_3$. Each of the shareholders say P_1, P_2, P_3 receives 2 out of the 3 shares, i.e. (r_2, r_3), (r_1, r_3) and (r_1, r_2) respectively. The secret s can be reconstructed if any 2 out of three shareholders combine their shares. However, a single shareholder will be unaware of one share of s and due to random distribution of shares, s remains private.

2.3.3 Garbling

A garbling scheme $\mathcal{G}$ is a technique used in MPC formalized by [BHR12]. and has been used by several works [JKO13, ZRE15, GLNP15]. A garbling scheme consists of four algorithms $(\mathsf{Gb}, \mathsf{En}, \mathsf{Ev}, \mathsf{De})$ where all but Gb are deterministic and are defined as follows:

$\mathsf{Gb}(1^\kappa, C) = (\mathsf{GC}, e, d)$ Gb takes as input the security parameter κ and the circuit C to be garbled, and outputs a garbled circuit GC, encoding information e and decoding information d.

$\mathsf{En}(x, e) = \mathbf{X}$ En encodes input x using e to output encoded input $\mathbf{X}$. We refer to $\mathbf{X}$ as encoded input or encoded labels interchangeably.

$\mathsf{Ev}(\mathsf{GC}, \mathbf{X}) = \mathbf{Y}$ Ev evaluates the garbled circuit GC on the encoded input $\mathbf{X}$ and produces the encoded output $\mathbf{Y}$.

$\mathsf{De}(\mathbf{Y}, d) = y$ The encoded output $\mathbf{Y}$ is decoded into the clear output y by running the De algorithm on $\mathbf{Y}$ and d.

A garbling scheme is required to satisfy the properties of *correctness*, *privacy*, *obliviousness* and *authenticity*. Informally, correctness ensures that a correctly garbled circuit returns the correct output of the underlying circuit when evaluated. Privacy ensures that encoded inputs don't leak information about the actual inputs. Obliviousness enforces that if the decoding information is withheld, then the garbled circuit evaluation doesn't leak information about the underlying clear values, be it the inputs, output or the intermediate wire values. Authenticity ensures that the evaluator learns only the encoded output corresponding to the actual circuit output.

Definition 1. *A garbling scheme* $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{Ev}, \mathsf{De})$ *is correct if for all input lengths* $n \leq \mathsf{poly}(\kappa)$*, circuit* $C : \{0,1\}^n \to \{0,1\}^m$ *and inputs* $x \in \{0,1\}^n$ *it holds that,*

$$\mathsf{Pr}[\mathsf{De}(\mathsf{Ev}(\boldsymbol{C}, En(x,e)), d) \neq C(x) : (\boldsymbol{C}, e, d) \leftarrow \mathsf{Gb}(1^\kappa, C)] \leq \mathsf{negl}(\kappa)$$

Definition 2. *A garbling scheme* $\mathcal{G}$ *is private if for all* $n \leq \mathsf{poly}(\kappa)$*, circuit* $C : \{0,1\}^n \to \{0,1\}^m$*, there exists a PPT simulator* $\mathcal{S}_{\mathbf{priv}}$ *such that for all* $x \in \{0,1\}^n$*, for all PPT adversary* $\mathcal{A}$ *the following distributions are computationally indistinguishable:*

- $\textsc{real}(C, x)$*: run* $(\boldsymbol{C}, e, d) \leftarrow \mathsf{Gb}(1^\kappa, C)$ *and output* $(\boldsymbol{C}, \mathsf{En}(x, e), d)$

- $\textsc{ideal}(C, C(x))$*: output* $(\boldsymbol{C}', \boldsymbol{X}, d') \leftarrow \mathcal{S}_{\mathbf{priv}}(1^\kappa, C, C(x))$

Definition 3. *A garbling scheme* $\mathcal{G}$ *is authentic if for all* $n \leq \mathsf{poly}(\kappa)$*, circuit* $C : \{0,1\}^n \to \{0,1\}^m$*, input* $x \in \{0,1\}^n$ *and for all PPT adversary* $\mathcal{A}$*, the following probability is* $\mathsf{negl}(\kappa)$*:*

$$\mathsf{Pr}\left(\begin{array}{l} \hat{\boldsymbol{Y}} \neq \mathsf{Ev}(\boldsymbol{C}, \boldsymbol{X}) \\ \wedge\, \mathsf{De}(\hat{\boldsymbol{Y}}, d) \neq \perp \end{array} : \begin{array}{r} \boldsymbol{X} = \mathsf{En}(x, e), (\boldsymbol{C}, e, d) \leftarrow \mathsf{Gb}(\kappa, C), \\ \hat{\boldsymbol{Y}} \leftarrow \mathcal{A}(\boldsymbol{C}, \boldsymbol{X}) \end{array}\right)$$

For our constructions, we use *projective* garbling schemes as defined in [BHR12].

Definition 4. *A projective garbling scheme is one where while garbling a circuit* $C : \{0,1\}^n \to \{0,1\}^m$*, the* e *has the form* $e = (e_i^0, e_i^1)_{i \in [n]}$*, and* $\boldsymbol{X}$ *for* $x = (x_i)_{i \in [n]}$ *can be interpreted as* $\mathbf{X} = \mathsf{En}(x, e) = (e_i^{x_i})_{i \in [n]}$.

2.3.4 Non-Interactive Commitment Scheme (NICOM)

A NICOM consists of two PPT algorithms (Com, Open) defined as:

- Com takes as input a security parameter κ, public parameter pp, a message x and random coins r, and outputs a commitment c and the corresponding opening information o.

- Open takes as input κ, pp, commitment c and corresponding opening information o, and outputs the message x.

Properties The commitment scheme must satisfy the properties: *hiding* (a commitment leaks no data about the underlying message), *binding* (a commitment cannot be opened to two different messages), and *correctness* (given the opening, Open should output the correct committed message).

- *Correctness.* For all messages $x \in \mathcal{M}$, $r \in \mathcal{R}$, pp, if $(c, o) \leftarrow \mathsf{Com}(x; r)$ then $\mathsf{Open}(c, o) = x$.

- *Hiding.* For all PPT adversaries $\mathcal{A}$, for all pp, $x, x' \in \mathcal{M}$, $|\mathsf{Pr}_{(c,o)\leftarrow\mathsf{Com}(x)}[\mathcal{A}(c) = 1] - \mathsf{Pr}_{(c,o)\leftarrow\mathsf{Com}(x')}[\mathcal{A}(c) = 1]| \leq \mathsf{negl}(\kappa)$.

- *Binding.* For all PPT adversaries $\mathcal{A}$, over uniform choice of pp and random coins of $\mathcal{A}$, probability that $\mathcal{A}$ outputs (c, o, o'), such that $\mathsf{Open}(c, o) \neq \mathsf{Open}(c, o')$ and $\perp \notin \{\mathsf{Open}(c, o), \mathsf{Open}(c, o')\}$, is negligible.

NICOM $(\mathsf{Com}, \mathsf{Open})$ is said to have strong binding if the binding property holds over all pp and not just over uniform choice of pp.

Instantiations In the random oracle model, strong commitment is defined as $\mathsf{Com}(x; r) = (c, o) = (\mathsf{H}(x||r), x||r)$. The pp can be empty. For implementation purpose, the random oracle is instantiated using a hash function (SHA-256). The following is an instantiation of a commitment scheme $(\mathsf{Com}, \mathsf{Open})$ with strong binding based on one-way permutation. Let $f : \{0,1\}^n \to \{0,1\}^n$ be a one-way permuation and $h : \{0,1\}^n \to \{0,1\}^n$ be its hardcore predicate. Commitment for a single bit x is defined as follows.

- $\mathsf{Com}(x; r)$: return $c = (f(r), x \oplus h(r))$ where $r \in \{0,1\}^n$, and $o = (r, x)$.

- $\mathsf{Open}(c, o = (r, x))$: return x if $c = (f(r), x \oplus h(r))$; else return $\perp$.

2.3.5 Equivocal Non-Interactive Commitment Scheme (eNICOM)

An Equivocal Non-Interactive Commitment Scheme (eNICOM) consists of four PPT algorithms $(\mathsf{eGen}, \mathsf{Equiv}, \mathsf{eCom}, \mathsf{eOpen})$. eCom and eOpen are as defined for NICOM. eGen and Equiv are used to equivocate a commitment to any desired message given a trapdoor. These are defined as follows.

- eGen takes as input κ and outputs a public parameter epp and a corresponding trapdoor t. This public parameter epp is used by both eCom and eOpen, and t is used for equivocation.
- Equiv takes as input a commitment c, its opening o, the desired message x and trapdoor t, and returns as output an opening o' such that $x \leftarrow \mathsf{eOpen}(\mathsf{epp}, c, o')$.

Properties An eNICOM scheme $(\mathsf{eCom}, \mathsf{eOpen})$ has the following properties.

- *Correctness* For all $x \in \mathcal{M}$, $r \in \mathcal{R}$, $(\mathsf{epp}, t) \leftarrow \mathsf{eGen}(1^\kappa)$, if $(c, o) \leftarrow \mathsf{eCom}(x; r)$ then $\mathsf{eOpen}(c, o) = x$.
- *Hiding* For all PPT adversaries $\mathcal{A}$, for all $(\mathsf{epp}, t) \leftarrow \mathsf{eGen}(1^\kappa)$, $x, x' \in \mathcal{M}$, the following is negligible, $|\mathsf{Pr}_{(c,o)\leftarrow\mathsf{eCom}(x)}[\mathcal{A}(c, o) = 1] - \mathsf{Pr}_{(c,o)\leftarrow\mathsf{eCom}(x'),o\leftarrow\mathsf{Equiv}(c,x,t)}[\mathcal{A}(c, o) = 1]| \leq \mathsf{negl}(\kappa)$
- *Binding* For all PPT $\mathcal{A}$, all $(\mathsf{epp}, t) \leftarrow \mathsf{eGen}(1^\kappa)$, probability that $\mathcal{A}$ outputs (c, o, o'), such that $\mathsf{eOpen}(c, o) \neq \mathsf{eOpen}(c, o')$ and $\perp \notin \{\mathsf{eOpen}(c, o), \mathsf{eOpen}(c, o')\}$, is negligible.

Instantiations The random oracle based commitment scheme defined before supports equivocation where $(\mathsf{epp}, t = (t_1, t_2))$ is empty. We rely on this for empirical purposes where random oracle is realized using hash function. In the plain model, the following is an equivocal bit commitment scheme from [CIO98] based on Naor's commitment scheme [Nao91] for single bit. Let $\mathsf{G} : \{0,1\}^n \rightarrow \{0,1\}^{4n}$ be a PRG.

- $\mathsf{eGen}(1^\kappa)$: set $(\mathsf{epp}, t_1, t_2) = ((\sigma, \mathsf{G}(r_0), \mathsf{G}(r_1)), r_0, r_1)$ where $\sigma = \mathsf{G}(r_0) \oplus \mathsf{G}(r_1)$.
- $\mathsf{eCom}(x; r)$: set $c = \mathsf{G}(r)$ if $x = 0$ else $c = \mathsf{G}(r) \oplus \sigma$; set $o = (r, x)$.
- $\mathsf{eOpen}(c, o = (r, x))$: return x if $c = \mathsf{G}(r) \oplus x \cdot \sigma$ (where $\cdot$ denotes multiplication by constant); else return $\perp$.
- $\mathsf{Equiv}(c = \mathsf{G}(r_0), \perp, x, (t_1, t_2))$: return $o = (r, x)$ where $r = t_1$ if $x = 0$ else $r = t_2$. Both t_1, t_2 are needed to perform equivocation.

Chapter 3

Building Blocks

Notations $\mathbf{P}$ denotes the set of all parties i.e. $\{P_1, P_2, P_3, P_4\}$. $\mathbf{P}_i$ and $\mathbf{P}_{ij}$ denote the sets $\mathbf{P} \setminus \{P_i\}$ and $\mathbf{P} \setminus \{P_i, P_j\}$ respectively. $\mathsf{ind}(S)$ denotes the set of indices of parties in set S.

Corrupt and Conflict Set Each party P_i locally maintains two sets: a corrupt (Corr_i) and a conflict (Con_i) set. The corrupt set Corr_i is populated with the identity of a party that is determined to be corrupt by P_i. $|\mathsf{Corr}_i| \leq 1$. The conflict set Con_i is populated with pairs of parties identified to be in conflict by P_i.

Input Commit routine We present a 2-round input-sharing routine $\mathsf{InputCommit}_i$ that enforces input consistency w.r.t. every party P_i's input x_i using RSS [CDI05a]. This technique of input sharing is a variant of the one in [BJPR18, IKKP15]. To recall, P_i splits its input x_i into three additive shares as $x_i = \oplus_{j \in \mathsf{ind}(\mathbf{P}_i)} x_{ij}$. As per RSS, each share x_{ij} is assigned to all parties except P_j. As shares are replicated, a corrupt dealer may create confusion by distributing inconsistent values of a particular share to the two shareholders. It is necessary for the honest shareholders to agree on all the shares at the end of the input-sharing routine. Such inconsistencies are handled by enforcing the dealer to commit to the shares and distribute the commitments to all parties with openings being sent only to the relevant shareholders. This allows to conclude on the majority commitment and its opening. To elaborate, in round 1, for each x_{ij}, commitment on x_{ij} is sent to all parties while the opening is sent to all but P_j. In round 2, all parties exchange the commitments received in the first round while only relevant shareholders exchange the openings. The commitment c_{ij} on the share x_{ij} is determined using a majority rule. If no majority exists among the commitments, then the dealer P_i is identified to be corrupt and a pre-determined default value is taken for P_i's input. If the commitments received from P_i (in round 1) and that forwarded by P_j (in round 2) mismatch, then P_i, P_j are added to the conflict set. Likewise, the corrupt and conflict set for openings are populated.

In our input-commit routine, we add the following additional rule to populate the corrupt set: if the commitment c_{ij} received from P_i does not turn out to be the commitment that is established as the majority, then P_i is added to the corrupt set. The definition below precisely defines the terms, *majority commitment*, *majority opening*, *committed share*. To ensure that a corrupt dealer does not modify her committed secret later, a strong NICOM is used. The formal $\mathsf{InputCommit}_i$ routine is described in Fig 3.1.

Input: Party P_i has input x_i.

Output: Each $P_k \in \mathbf{P}_i$ outputs $(\{c_{ij}\}_{j\in\mathsf{ind}(\mathbf{P}_i)}, \{o_{ij}, x_{ij}\}_{j\in\mathsf{ind}(\mathbf{P}_{ik})}, \mathsf{Corr}_k, \mathsf{Con}_k)$. $\{c_{ij}, o_{ij}\}$ respectively denote commitment and opening of share x_{ij}.

Primitives: A strong NICOM $(\mathsf{Com}, \mathsf{Open})$.

Round 1: For $j \in \mathsf{ind}(\mathbf{P}_i)$, P_i splits her input as $x_i = \oplus_j x_{ij}$, samples pp and computes $(c_{ij}, o_{ij}) \leftarrow \mathsf{Com}(\mathsf{pp}, x_{ij})$. P_i sends (pp, c_{ij}) to $\mathbf{P}_i$ and o_{ij} to $\mathbf{P}_{ij}$.

Local Computation by $P_k \in \mathbf{P}_{ij}$: Set $\mathsf{Corr}_k = \{P_i\}$ and $c_{ij} = o_{ij} = \perp$ if $\mathsf{Open}(\mathsf{pp}, c_{ij}, o_{ij}) = \perp$ for any o_{ij} received.

Round 2: The following is done for each $j \in \mathsf{ind}(\mathbf{P}_i)$:
- $P_k \in \mathbf{P}_i$ forwards (pp, c_{ij}) to other parties in $\mathbf{P}_i$.
- $P_k \in \mathbf{P}_{ij}$ forwards o_{ij} to the other party in $\mathbf{P}_{ij}$.

Local Computation for each $j \in \mathsf{ind}(\mathbf{P}_i)$:
- $P_k \in \mathbf{P}_i$ adds (P_i, P_l) to Con_k if received two different copies of (pp, c_{ij}) from P_i in Round 1 and P_l in Round 2.
- $P_k \in \mathbf{P}_i$ sets $\mathsf{Corr}_k = \{P_i\}$ if no honest majority exists among the three versions of (pp, c_{ij}) sent by P_i in Round 1 and forwarded by $\mathbf{P}_{ik}$ in Round 2. Also, P_k sets default commitments for each $x_{im}, m \in \mathsf{ind}(\mathbf{P}_i)$ and pre-agreed default share-values for x_{im}. (This is done to ensure x_i is set to a pre-agreed default value.)
- Otherwise, $P_k \in \mathbf{P}_i$ sets (pp, c_{ij}) as the majority commitment and $P_k \in \mathbf{P}_{ij}$ further sets the respective opening o_{ij} as the majority opening to obtain the committed share $x_{ij} = \mathsf{Open}(\mathsf{pp}, c_{ij}, o_{ij})$.
- $P_k \in \mathbf{P}_i$ sets $\mathsf{Corr}_k = \{P_i\}$ if (pp, c_{ij}) received from P_i in Round 1 does not turn out to be the majority commitment.
- $P_k \in \mathbf{P}_{ij}$ adds P_l to Corr_k if the commitment-opening pair (c_{ij}, o_{ij}) forwarded by P_l in round 2 is invalid $(\mathsf{Open}(\mathsf{pp}, c_{ij}, o_{ij}) = \perp)$.

Figure 3.1: Input Commit routine $\mathsf{InputCommit}_i()$

Definition 5. *A share x_{ij} is said to be **committed** after $\mathsf{InputCommit}_i$ if every honest $P_k \in$*

$\mathsf{ind}(\mathbf{P}_i)$ *holds* c_{ij} *such that every honest* $P_k \in \mathbf{P}_{ij}$ *holds an opening* o_{ij} *that opens* c_{ij} *to* x_{ij}*. The corresponding commitment (to* x_{ij}*) is called the* ***majority commitment*** *and the corresponding opening is called the* ***majority opening****.*

Lemma 1. *A party either commits to unique input or is publicly identified to be corrupt.*

Proof. A corrupt P_i, for its share x_{ij}, can send same (pp, c_{ij}) to at least two parties in $\mathbf{P}_i$ and a corresponding valid opening to at least one of $\mathbf{P}_{ij}$ in which case (pp, c_{ij}) will be chosen as majority commitment and x_{ij} as the committed share. The strong binding of NICOM ensures that the share is unique. Otherwise, no honest majority will exist for x_{ij} at the end of round 2 and P_i is added to the corrupt set of all honest parties with a default value taken for x_{ij}. For honest P_i, each share x_{ij} and thus x_i chosen by P_i remains committed, irrespective of the corrupt party's behaviour in the $\mathsf{InputCommit}_i$, as there is only one corrupt party in $\mathbf{P}_i$. □

Lemma 2. *A corrupt party* P_i *either belongs to* Con_j *or* Corr_j *of each honest party* P_j *if it behaves adversarially in* $\mathsf{InputCommit}_i$*.*

Proof. A corrupt party P_i, for its share x_{ij}, can send (a) Two same and one different copy of (pp, c_{ij}) with valid opening to one party in $\mathbf{P}_{ij}$. (b) All 3 different copies of (pp, c_{ij}). (c) Two same and one different copy of (pp, c_{ij}) with no valid opening to either party in $\mathbf{P}_{ij}$. In case (a), the recipients of a different copy and invalid opening add P_i to their corrupt set and the remaining honest parties add, to their conflict set, a pair consisting of P_i. In case (b) and case (c), all parties add P_i to their corrupt set as no majority commitment will be established at the end of round 2. □

Chapter 4

Fairness in 3 rounds

Relaxing the security notion from GOD to fairness, we present an efficient 3 round 4PC that achieves fairness against 1 active corruption, in a minimal network of pairwise-private channels at the expense of 2GCs and improve upon the state-of-the-art efficient protocols of [BJPR18] that achieve GOD in 5 and 4 roun'ds with 1 GC and 2 GC respectively. Building upon [BJPR18], we add several new techniques to minimize the round complexity and achieve fairness in a highly efficient manner.

4.1 The construction

We adopt a GC based approach [Yao82] with two garblers (P_1, P_2) and two evaluators (P_3, P_4) to leverage the guarantee of at least one honest evaluator. We use two garbling instances, both with $\{P_1, P_2\}$ as garblers with P_3 and P_4 as respective evaluators. We use semi-honest GC and achieve security against a malicious garbler by allowing the garblers to generate the same GC (with shared randomness) as in [MRZ15]. The 2 rounds of InputCommit routine aid in input distribution. The garblers exchange randomness used to generate the GC in round 1. Round 2 of InputCommit is overlapped with sending of GC (without decoding information) and encoded inputs needed for evaluation to save rounds as in [BJPR18]. A simple check of equality suffices (due to the presence of an honest garbler) to determine the correctness of GC. Finally, round 3 is used to exchange the decoding information and encoded output $\mathbf{Y}$ for fair output computation. We summarize our techniques below.

On the transfer of encoding information The RSS sharing of input ensures that two parties collectively hold all input shares. Hence, it suffices for the garblers to send the encoded labels for the shares they own to each evaluator. To enable an honest evaluator to detect incorrect / inconsistent labels sent by a potentially corrupt garbler, we adopt the standard

commit-then-open approach of [MRZ15, BJPR18] where the garblers perform commitments on the encoding information (e) in a permuted fashion and send openings for the shares they own to the evaluator. This suffices since, for an input share not held by the evaluator, it is available with both garblers and thus, the evaluator can verify if both the openings received for such a share are the same. The use of permutation here further ensures that the evaluator does not learn the underlying value of the encoded label that she has the opening for. However, for the input shares held by the evaluator, the permutation is also revealed to the evaluator to help verify if the correct opening has been received. This method of communication brings along the following concern for the evaluator's (say P_3) input share which both the garblers hold (x_{34}). A corrupt P_3 might disseminate inconsistent x_{34} to the garblers in round 1 of $\mathsf{InputCommit}_3$ which leads to them sending different encoded labels for x_{34} and thus enabling P_3 to perform evaluation on multiple inputs. Despite withholding decoding information, this still allows P_3 to compare the two resulting encoded outputs and conclude if they are the same for two different input shares. We tackle this as in [BJPR18] by allowing only one garbler to send the opening for the input shares (x_{34} in x_3) of the evaluator (P_3) while for shares corresponding to inputs of other parties, each garbler sends the openings for the share she owns to P_3.

On Evaluation and Output Computation During evaluation, the following invariant is maintained by an honest evaluator: evaluation of GC is performed only if all the received encoded labels correspond to the *committed* input shares as per Definition 5. Else the corrupt and conflict sets are populated accordingly. This ensures that the output, if computed corresponds to a unique set of inputs. A corrupt evaluator P_4 can trivially violate fairness by deliberately not revealing the encoded output ($\mathbf{Y}$) to the remaining parties on successful evaluation. This is tackled robustly with the use of i) two evaluators to ensure the presence of at least one honest evaluator and ii) *oblivious* garbling along with withholding of the decoding information (d) by garblers until they are convinced that evaluation is indeed performed on *committed* inputs. Specifically, we restrict a garbler to send d to an evaluator only if the evaluator is neither in her corrupt set nor identified to be in conflict with the co-garbler. This ensures that d is revealed to an evaluator only if the invariant is maintained. Further, point (i) ensures that one (honest) of the two evaluators reveals $\mathbf{Y}$ to enable everyone to compute the output.

4.2 Our Techniques

Splitting of decoding information The above restriction for revealing d gives rise to another issue. A corrupt P_1 can misbehave in the $\mathsf{InputCommit}$ routine and send encoded labels in a way that all checks pass for honest P_3 who is able to evaluate on committed inputs but

P_4 is unable to; resulting in P_4 populating either Con_4 with (P_1, P_2) or Corr_4 with P_1. Further, P_1 also ensures that (P_1, P_3) are in Con_2 by misbehaving in $\mathsf{InputCommit}_3$. As a result, despite evaluating on committed inputs and sending $\mathbf{Y}$ to all, P_3 gets d neither from corrupt P_1 nor from honest P_2, but the corrupt P_1 has learnt the output violating fairness. From P_2's perspective, this scenario is indistinguishable to the case when P_3 is indeed corrupt and evaluated on inputs that do not correspond to committed input shares. To tackle this, we use the deduction that P_4 has identified P_3 to be honest. With this knowledge, we enforce the garblers to split d (say for GC that P_3 evaluates) as $d = d_3 \oplus d_4$ and commit on these shares for consistency. The garblers send commitments on both the shares (for equality check) to both evaluators, but the openings for d_3 and d_4 are sent only to P_3 and P_4 respectively. Thus, to enable P_3 to compute the output, P_4 reveals the opening of share d_4 (of d) she holds to P_3 when P_4 herself cannot evaluate the GC but identifies P_3 to be honest. Also, P_3 sends the opening of commitment on d_3 along with $\mathbf{Y}$ to P_4 to let P_4 compute the output. Lastly, to convince the honest P_2 to accept $\mathbf{Y}$ sent by honest P_3, we enable an evaluator (here P_4) to additionally send a signal to the garblers to indicate the honesty of her co-evaluator (here P_3). This signal suffices for P_2 to accept $\mathbf{Y}$ sent by P_3 since the signal confirmation was received from P_4 known to be honest to P_2 (as $\mathsf{Con}_2 = \{(P_1, P_3)\}$).

Splitting openings of encoding labels With the above solution of splitting of d and signaling in place, a corrupt P_1 can still violate fairness as: P_1 misbehaves in $\mathsf{InputCommit}_3$ (for x_{32} by sending incorrect c_{32} to P_2 and P_4) s.t. the invariant of evaluation on committed inputs holds for P_3 but not P_4. Both P_2, P_4 put (P_1, P_3) in conflict and consequently, honest P_2, P_4 neither reveal the decoding information to P_3 nor accept $\mathbf{Y}$ computed by P_3, whereas P_1 is successful in obtaining the output. Since it appears that no solution other than making P_4 evaluate works in this case, we provide P_3 a chance to help P_4 obtain the encoded labels that correspond to the committed shares (x_{31}, x_{32}) of P_3. To elaborate, we split the openings of encoding labels corresponding to the input-shares of an evaluator (here P_3) between P_3 and the garblers. This modification is not required for share x_{34} as it is with both garblers and at least one of them would send an opening of the encoded label corresponding to the *committed* x_{34}, as P_3 is honest (Lemma 1). P_1, P_2 redundantly send their additive shares of openings (for *both* labels) to P_4 in round 2 in addition to sending the *entire* opening (of *one* corresponding label) for the input shares x_{31}, x_{32}. P_3 also sends her opening share for label corresponding to x_{31}, x_{32} to P_4. Since P_4 holds x_{32} as per RSS, P_4 can now add the shares of the opening sent by P_3 and the corresponding share (out of the two) sent by P_2 (as P_2 is identified to be honest by P_4) to determine if the resulting value forms a valid opening for the encoded label corresponding to x_{32}. If so, P_4 too evaluates the GC and the evaluators help each other to compute the output

while P_2 accepts the output from P_4. However, generating the two additive shares of these openings poses a challenge since the entire GC and relevant data is generated using the randomness shared by the garblers in round 1 and also there are not enough rounds available before the communication of encoded labels, for the garblers and the evaluator involved to arrive at a consensus about the consistency of the opening shares. Hence, we make the evaluator P_3 sample random shares for the encoding information corresponding to her input shares x_{31}, x_{32} and send the same to the garblers in round 1. The garblers use the random shares sent by P_3 to appropriately compute the second share of each opening of the labels. Since, these openings are used by P_4 only in the case when the invariant of evaluation on committed inputs is violated due to P_3's input shares x_{31}, x_{32}, it suffices to not have agreement of the random shares received by the garblers from P_3 amongst themselves. In the above case, since P_4 knows P_2 is honest, P_4 uses the opening shares sent by P_2 alone along with P_3's and if the addition of their shares still leads to an opening that is invalid, then P_4 can conclude that P_3 is corrupt and case (ii) follows. Thus, splitting of openings corresponding to the encoding information for evaluator's input coupled with splitting of d and signalling suffices to handle fairness violation by a corrupt garbler.

Why 2 different GCs? There arises a subtle case of evaluation on multiple inputs by a corrupt evaluator due to the inclusion of all the above techniques. A corrupt P_4 may send some x_{42} to P_1 in round 1 that does not correspond to the *committed* input share x_{42} committed at the end of round 2, thus violating the invariant of evaluation on committed inputs. This causes honest P_2, P_3 to put (P_1, P_4) in conflict and P_1 to add P_4 to Corr_4 at the end of round 2. However, P_4 helps P_3 obtain the encoded label for x_{42} with the technique of splitting the openings and enables P_3 to evaluate the GC. Thus, P_3 evaluates the GC on committed inputs whereas the corrupt P_4 evaluates the GC on whatever labels obtained from P_1. In round 3, P_3 sends $\mathbf{Y}$ and opening for d_3 to P_4 who now can decode both, (the encoded output computed by P_4 and the $\mathbf{Y}$ sent by P_3) and obtain output on multiple inputs. To resolve this, we maintain that a garbler who evaluated on non-committed inputs never gets access to the corresponding decoding information. This is enforced by following 2 steps: (a) The garblers sample two different randomness (one for P_3, one for P_4) and generate two unique GCs and send them to P_3 and P_4 respectively. This results in generating shares of decoding information for each GC and exactly one share of each decoding information is revealed to an evaluator in round 2. (b) When an evaluator, say P_3 (in above example) successfully evaluates his GC (possibly with P_4's aid), he gives the decoding information that P_4 misses only corr. to P_3's GC. We use equivocal commitments to commit on the shares of d to tackle a technical aspect in the simulation proof. This completes the protocol description. The formal details of one garbling instance and the

final protocol appear in Figs 4.1 and 4.2 respectively. We extend this fair protocol to achieve GOD in Chapter 5. We denote the steps relevant to GOD alone with **bold blue color**.

Common Inputs: Circuit $C(x_1, x_2, x_3, x_4)$ that computes function $f(\oplus_{j\in\mathsf{ind}(\mathbf{P}_1)}x_{1j}, \oplus_{j\in\mathsf{ind}(\mathbf{P}_2)}x_{2j}, \oplus_{j\in\mathsf{ind}(\mathbf{P}_3)}x_{3j}, \oplus_{j\in\mathsf{ind}(\mathbf{P}_4)}x_{4j})$ with each input, their shares and output of the form $\{0,1\}$ (for simplicity).

Notations: We denote the evaluator receiving the GC in the routine as P_v and the remaining evaluator as P_h (helper) and $v, h \in \{3,4\}$. We denote the additive sharing of an opening o with $[\cdot]$ notation. The first share is denoted as $[o]_0$ and the second share is denoted as $[o]_1$.

Output: Evaluator P_v outputs $\mathbf{B}_v^g/\perp$ and P_h outputs $\mathbf{B}'^g_v/\perp$.

Primitives: Garbling $(\mathsf{Gb}, \mathsf{En}, \mathsf{Ev}, \mathsf{De})$, a PRG G, a collision resistant hash H, an eNICOM $(\mathsf{eCom}, \mathsf{eOpen})$, a strong NICOM $(\mathsf{Com}, \mathsf{Open})$.

Round 1:

- P_1 samples random seed $r \leftarrow \{0,1\}^\kappa$ for G and sends to P_2.
- P_1 samples share epp_1 for eNICOM and sends epp_1 to P_2. Similarly, P_2 samples share epp_2 and sends epp_2 to P_1.
- P_h samples random $\{[o^b_{h1}]_0, [o^b_{h2}]_0\}_{b\in\{0,1\}}$ for encoding labels belonging to shares x_{h1}, x_{h2}, sends $\{[o^b_{h1}]_0, [o^b_{h2}]_0\}_{b\in\{0,1\}}$ to P_1, P_2.

Round 2:

- Each garbler, $P_g, g \in [2]$ computes $s = \mathsf{G}(r)$, $\mathsf{epp} = \mathsf{epp}_1 \oplus \mathsf{epp}_2$ and does the following using randomness s:
  - Garble the circuit C as $(\mathsf{GC}, e, d) \leftarrow \mathsf{Gb}(1^\kappa, C)$. Sample pp derived from s for strong NICOM.
  - Let $p_{ij} \in_R \{0,1\}$ denote the permutation bit for input wire corresponding to P_i's input share x_{ij}, $i \in [4]$, $j \in \mathsf{ind}(\mathbf{P}_i)$. Set $p_i \leftarrow ||_{j\in\mathsf{ind}(\mathbf{P}_i)} p_{ij}$. Set $p \leftarrow ||_{i\in[4]} p_i$.
  - For $i \in [4], j \in \mathsf{ind}(\mathbf{P}_i)$, let $\{e^0_{ij}, e^1_{ij}\}$ denote encoding labels belonging to the j^{th} share of P_i's input i.e. x_{ij}. For $b \in \{0,1\}$, compute commitments as: $(c^b_{ij}, o^b_{ij}) \leftarrow \mathsf{Com}(\mathsf{pp}, e^{p_{ij}\oplus b}_{ij})$.
  - For $\{o^b_{hi}\}_{b\in\{0,1\}, i\in[2]}$ belonging to input shares x_{hi}, set $[o^b_{hi}]_1 = o^b_{hi} \oplus [o^b_{hi}]_0$.
  - Split the decoding information d as $d = d_3 + d_4$. Compute: $(\mathsf{c}_3, \mathsf{o}_3) \leftarrow \mathsf{eCom}(\mathsf{epp}, d_3)$, $(\mathsf{c}_4, \mathsf{o}_4) \leftarrow \mathsf{eCom}(\mathsf{epp}, d_4)$.
  - Let $\mathbf{p}_v = \{p_{ij}\}_{i\in[4], j\in\mathsf{ind}(\mathbf{P}_{iv})}$ denote the set of permutation bits corresponding to the shares possessed by P_v. Set $\mathbf{B}^g_v = \{\mathsf{GC}, \{c^b_{ij}\}_{i\in[4], j\in\mathsf{ind}(\mathbf{P}_i), b\in\{0,1\}}, \mathsf{c}_3, \mathsf{c}_4, \mathsf{o}_3, \mathbf{p}_v\}$ and $\mathbf{B}'^g_v = \{\mathsf{c}_3, \mathsf{c}_4, \mathsf{o}_4\}$.
- P_1 sends $\mathbf{B}_v$ and P_2 sends $\mathsf{H}(\mathbf{B}_v)$ to P_v. P_v sets $\mathsf{Con}_v = \mathsf{Con}_v \cup \{(P_1, P_2)\}$ if inconsistent copies are received. **Set conflict** $= 1$
- P_1 sends $\mathbf{B}'_v$ to P_h and P_2 sends $\mathsf{H}(\mathbf{B}'_v)$ to P_h. If the copies are inconsistent, P_h sets $\mathsf{Con}_h = \mathsf{Con}_h \cup \{(P_1, P_2)\}$. **Set conflict** $= 1$.

Figure 4.1: Garbling routine Garble_{vh}

Inputs and Output: Party P_i has input $x_i, i \in [4]$. Each party outputs $y = f(x_1, x_2, x_3, x_4)$ or $\perp$.

Common Inputs: Circuit $C(x_1, x_2, x_3, x_4)$ that computes function $f(\oplus_{j\in\mathsf{ind}(\mathbf{P}_1)}x_{1j}, \oplus_{j\in\mathsf{ind}(\mathbf{P}_2)}x_{2j}, \oplus_{j\in\mathsf{ind}(\mathbf{P}_3)}x_{3j}, \oplus_{j\in\mathsf{ind}(\mathbf{P}_4)}x_{4j})$ with each input, their shares and output taken from $\{0,1\}$ (instead of $\{0,1\}^\ell$ for simplicity).

Primitives: A strong (NICOM) $(\mathsf{Com}, \mathsf{Open})$, a collision resistant hash H.

Round 1: Round 1 of $\mathsf{InputCommit}_i()$ (Fig 3.1) is run for each $i \in [4]$. Round 1 of Garble_{34} (Fig 4.1) is run. Analogously round 1 of Garble_{43} is run.

Round 2: Round 2 of $\mathsf{InputCommit}_i()$ (Fig 3.1) is run for each $i \in [4]$. Besides, the following is done:

- Round 2 of Garble_{34} (Fig 4.1) is run. Analogously round 2 of Garble_{43} is run. Denote the sets obtained by P_3 as $\mathbf{B}_3 = \left\{\mathsf{GC}, \{c^b_{ij}\}_{i\in[4],j\in\mathsf{ind}(\mathbf{P}_i),b\in\{0,1\}}, \mathsf{c}_3, \mathsf{c}_4, \mathsf{o}_3, \mathbf{p}_3\right\}$ and $\mathbf{B'}_4 = \{\tilde{\mathsf{c}}_3, \tilde{\mathsf{c}}_4, \tilde{\mathsf{o}}_3\}$ and the sets obtained by P_4 as $\mathbf{B}_4 = \left\{\tilde{\mathsf{GC}}, \{\tilde{c}^b_{ij}\}_{i\in[4],j\in\mathsf{ind}(\mathbf{P}_i),b\in\{0,1\}}, \tilde{\mathsf{c}}_3, \tilde{\mathsf{c}}_4, \tilde{\mathsf{o}}_4, \mathbf{p}_4\right\}$ and $\mathbf{B'}_3 = \{\mathsf{c}_3, \mathsf{c}_4, \mathsf{o}_4\}$.

Communication of encoded labels:

- Let $m_{ij} = x_{ij} \oplus p_{ij}$ where $i \in [4], j \in \mathsf{ind}(\mathbf{P}_i)$. If $\mathsf{Corr}_g = \varnothing$, then $P_g, g \in [2]$ sends openings for input wires corresponding to the input shares $\{x_{ij}\}_{i\in[4],j\in\mathsf{ind}(\mathbf{P}_{ig})}$ to evaluator $P_v, v \in \{3,4\}$. P_g also sends $\{m_{ij}\}_{i\in[4],j\in\mathsf{ind}(\mathbf{P}_{ig})}$ to P_v. The common share of P_v's input (x_{34} for P_3, x_{43} for P_4) is opened by exactly one garbler. The opening corresponding to x_{34} is sent by P_1 to P_3 for GC and that corresponding to x_{43} is sent by P_2 to P_4 for $\tilde{\mathsf{GC}}$.
- If $\mathsf{Corr}_g = \varnothing$, then $P_g, g \in [2]$ also sends $\{[o^b_{41}]_1, [o^b_{42}]_1\}_{b\in\{0,1\}}$ for input wires belonging to shares x_{41}, x_{42} to P_3. Similarly, P_g sends $\{[\tilde{o}^b_{31}]_1, [\tilde{o}^b_{32}]_1\}_{b\in\{0,1\}}$ for input wires belonging to shares x_{31}, x_{32} to P_4.
- If $\mathsf{Corr}_4 = \varnothing$, P_4 sends $\{[o^{x_{41}}_{41}]_0, [o^{x_{42}}_{42}]_0\}$ for input wires belonging to shares x_{41}, x_{42} to P_3. Similarly, if $\mathsf{Corr}_3 = \varnothing$, P_3 sends $\{[\tilde{o}^{x_{31}}_{31}]_0, [\tilde{o}^{x_{32}}_{32}]_0\}$ for input wires belonging to shares x_{31}, x_{32} to P_4.

Local computation by $P_v, v \in \{3,4\}$:

- If $\mathsf{Con}_v = \mathsf{Corr}_v = \varnothing$ (no conflict or corrupt detected so far): Add P_g to Corr_v if the indices $\{m_{ij}\}_{i\in[4],j\in\mathsf{ind}(\mathbf{P}_{iv})}$ computed using its version of x_{ij} and p_{ij} received from garblers mismatches with the copy sent by P_g. Also, add P_g to Corr_v if any of the openings sent by P_g is invalid (results in $\perp$). **Set corrupt$_g = 1$, honest$_k = 1$ for $k = [2] \setminus \{g\}$.** For the input shares not held by P_v, add (P_1, P_2) to Con_v if $\{m_{iv}\}_{i\in[4]}$ sent by P_1 and P_2 are not the same. **Set conflict $= 1$.** Else, compute $e_{iv} \leftarrow \mathsf{Open}(\mathsf{pp}, c^{m_{iv}}_{iv}, o^{m_{iv}}_{iv})$.
- For $P_v = P_3$ and $g \in [2]$, if the opening $o^{m_{4j}}_{4j}$ sent by P_g does not correspond to the committed share $x_{4j}, j \in [2] \setminus \{g\}$, then add (P_g, P_4) to Con_3 and compute $o^{x_{4j}}_{4j} = [o^{x_{4j}}_{4j}]_0 \oplus [o^{x_{4j}}_{4j}]_1$ where $[o^{x_{4j}}_{4j}]_1$ is the copy sent by P_j (honest). If the opening is still invalid (results in $\perp$), then add P_4 to Corr_3. **Set corrupt$_4 = 1$, honest$_j = 1$** Else compute $e_{4j} \leftarrow \mathsf{Open}(\mathsf{pp}, c^{m_{4j}}_{4j}, o^{x_{4j}}_{4j})$. Similar steps are done

for $P_v = P_4$.

If all encoded labels received correspond to the committed input shares, then P_3 sets $\mathbf{X}$ to be the encoded input. P_3 computes $\mathbf{Y} = \mathsf{Ev}(\mathsf{GC}, \mathbf{X})$. Similarly, P_4 sets $\tilde{\mathbf{X}}$ to be the encoded input. P_4 computes $\tilde{\mathbf{Y}} = \mathsf{Ev}(\tilde{\mathsf{GC}}, \tilde{\mathbf{X}})$.

Round 3:

- If obtained $\mathbf{Y}$, P_3 sends $\mathbf{Y}$ to the garblers and $(\mathbf{Y}, \mathsf{o}_3)$ to P_4. Similar steps are performed by P_4 if obtained $\tilde{\mathbf{Y}}$.
- If not computed $\mathbf{Y}$ and $P_4 \notin \mathsf{Con}_3 \cup \mathsf{Corr}_3$, then P_3 sends (P_4, HSig) to the garblers and $\tilde{\mathsf{o}}_3$ to P_4. Similarly, if P_4 has not computed $\tilde{\mathbf{Y}}$ and $P_3 \notin \mathsf{Con}_4 \cup \mathsf{Corr}_4$, then P_4 sends (P_3, HSig) to the garblers and o_4 to P_3.
- For $g \in [2], j \in [2] \setminus \{g\}$, if $P_3 \notin \mathsf{Corr}_g$ and $(P_j, P_3) \notin \mathsf{Con}_g$, then P_g sends o_4 to P_3. Similarly, if $P_4 \notin \mathsf{Corr}_g$ and $(P_j, P_4) \notin \mathsf{Con}_g$, P_g sends $\tilde{\mathsf{o}}_3$ to P_4.

Local Computation by $P_g, g \in [2]$:

- If received $\mathbf{Y}$ from P_3 and $\tilde{\mathbf{Y}}$ from P_4, compute $y = \mathsf{De}(\mathbf{Y}, d)$.
- Else if received $\mathbf{Y}$ from P_3 and $\mathsf{HSig}(P_3)$ from P_4, compute $y = \mathsf{De}(\mathbf{Y}, d)$. Similarly, if received $\tilde{\mathbf{Y}}$ from P_4 and $\mathsf{HSig}(P_4)$ from P_3, compute $y = \mathsf{De}(\tilde{\mathbf{Y}}, \tilde{d})$.
- Else if received $\mathbf{Y}$ (or $\tilde{\mathbf{Y}}$) received from P_3 (or P_4) and $(P_3, P_4) \in \mathsf{Con}_g$, compute $y = \mathsf{De}(\mathbf{Y}, d)$ ($y = \mathsf{De}(\tilde{\mathbf{Y}}, \tilde{d})$).
- Else if received $\mathbf{Y}$ from P_3 and $P_3 \notin \mathsf{Con}_g \cup \mathsf{Corr}_g$, compute $y = \mathsf{De}(\mathbf{Y}, d)$. Likewise, if got $\tilde{\mathbf{Y}}$ from P_4 and $P_4 \notin \mathsf{Con}_g \cup \mathsf{Corr}_g$, compute $y = \mathsf{De}(\tilde{\mathbf{Y}}, \tilde{d})$.

Local Computation by P_3: If sent $\mathbf{Y}$ and received o_4 from one of P_1, P_2, P_4, then compute $d_4 \leftarrow \mathsf{eOpen}(\mathsf{epp}, \mathsf{c}_4, \mathsf{o}_4)$ and $d = d_3 \oplus d_4$ and $y = \mathsf{De}(\mathbf{Y}, d)$. Else if received $(\tilde{\mathbf{Y}}, \tilde{\mathsf{o}_4})$ from P_4 and $P_4 \notin \mathsf{Con}_3 \cup \mathsf{Corr}_3$, then compute $\tilde{d_4} \leftarrow \mathsf{eOpen}(\mathsf{epp}, \tilde{\mathsf{c}}_4, \tilde{\mathsf{o}}_4)$ and $\tilde{d} = \tilde{d_3} \oplus \tilde{d_4}$ and $y = \mathsf{De}(\tilde{\mathbf{Y}}, \tilde{d})$.

Local Computation by P_4: If sent $\tilde{\mathbf{Y}}$ and received $\tilde{\mathsf{o}}_3$ from one of P_1, P_2, P_3, then compute $\tilde{d_3} \leftarrow \mathsf{eOpen}(\mathsf{epp}, \tilde{\mathsf{c}}_4, \tilde{\mathsf{o}}_4)$ and $\tilde{d} = \tilde{d_3} \oplus \tilde{d_4}$ and $y = \mathsf{De}(\tilde{\mathbf{Y}}, \tilde{d})$. Else if received $(\mathbf{Y}, \mathsf{o}_3)$ from P_3 and $P_3 \notin \mathsf{Con}_4 \cup \mathsf{Corr}_4$, then compute $d_3 \leftarrow \mathsf{eOpen}(\mathsf{epp}, \mathsf{c}_3, \mathsf{o}_3)$ and $d = d_3 \oplus d_4$ and $y = \mathsf{De}(\mathbf{Y}, d)$.

Figure 4.2: 3-round Fair Protocol 3RFair

4.3 Optimizations

The following optimizations are adopted to boost efficiency. The garblers P_1 sends the common information $\mathbf{B}_3$ to P_3 and P_2 sends $\mathbf{B}_4$ to P_4. However, P_1 sends $\mathsf{H}(\mathbf{B}_4)$ to P_4 and P_2 sends $\mathsf{H}(\mathbf{B}_3)$ to P_3. This optimization reduces communication and in turn improves the latency of communication of the garbled circuit. A similar technique is used to communicate $\mathbf{B'}_3$ and $\mathbf{B'}_4$. We also improve communication by performing equivocal commitments on the hash

of the shares of the decoding information. For empirical purposes, we instantiate equivocal commitments with hash function since the hash function can be visualized as a programmable random oracle.

4.4 Correctness and Security

Theorem 1. *Assuming one-way permutations exist, the protocol* 3RFair *(Fig 4.2) securely realizes the functionality* $\mathcal{F}_{\mathsf{fair}}$ *(Fig 2.2) against one malicious corruption in the standard model.*

4.4.1 Correctness

This sections proves correctness which is established via a series of lemmas as follows:

Lemma 3. *No two honest parties belong to the conflict set* Con_i *of an honest party* P_i*.*

Proof. An honest P_i populates its conflict set with (P_j, P_k) only if: (i). In $\mathsf{InputCommit}()$ routine, if the version of commitments for any share received from P_j, P_k mismatch. (ii). When P_j, P_k are garblers and (iia) the received $\mathbf{B}_i, \mathbf{B'}_i$ or (iib). when $\mathsf{Con}_i = \varnothing$ by the end of round 2, but the m_{lj}'s received for $l \in [4], j \in \mathsf{ind}(\mathbf{P}_i)$ mismatch. Cases (i),(iia) cannot happen for an honest P_j, P_k. However, for case (iib), it must be the case that P_j, P_k were in agreement about the corrupt party's share. Else, P_i would put one of P_j, P_k in the conflict pair with the corrupt party, but in either case, $(P_j, P_k) \notin \mathsf{Con}_i$ and contradicts the assumption of $\mathsf{Con}_i = \varnothing$ in case (iib). □

Lemma 4. *For a pair of honest parties* P_i, P_j*, the following holds:* $P_i \notin \mathsf{Corr}_j$*.*

Proof. An honest P_i adds P_j to Corr_i only if: (i). In $\mathsf{InputCommit}_j$, there was no majority commitment or corresponding valid opening for some x_{jk}. (ii). If $\mathsf{Corr}_i = \varnothing$ at the end of round 2, but P_i as an evaluator did not evaluate, as no encoded labels were available for committed x_{j2} (P_j as co-evaluator). (iii). P_j as garbler sent invalid openings ($\perp$) for encoded labels input shares held by her or sends labels inconsistent with her messages in round 1 of $\mathsf{InputCommit}$. Since neither of the above cases are possible for an honest P_j, the lemma holds. □

Lemma 5. *An honest evaluator evaluates the GC on committed inputs alone.*

Proof. An honest evaluator P_i evaluates the GC only if all encoded labels received correspond to the committed input shares. This implies that the corrupt party P_j commits to its input in round 1 of $\mathsf{InputCommit}_j$ and sends encoded labels wrt them. Further, a corrupt garbler P_j is forced to send valid openings for encoded labels corresponding to committed input shares belonging to her. Else P_j will be identified to be corrupt and the invariant of evaluation on

committed input shares fails. However, for the co-evaluator P_k's input share owned by P_j, even if P_j sends invalid/non-committed openings, P_k or the honest co-garbler (for x_{ki}) will help P_i obtain the labels for committed share. Hence the lemma. □

Lemma 6. *A corrupt evaluator does not receive the decoding information for a circuit evaluated on inputs that are not committed.*

Proof. For a corrupt evaluator P_i will receive the decoding information only if (i). $P_i \notin \mathsf{Corr}_j$ for garbler P_j and P_i is not in conflict with her co-garbler. (ii). If the co-evaluator P_k has garblers in conflict or one of the garblers in corrupt set. If P_i evaluates on her input that is not committed (honest parties' inputs are committed by the end of round 1) , case (i) will not hold since, at the end $\mathsf{InputCommit}_i$, an honest garbler P_j will populate either Corr_j or Con_i with P_i and co-garbler. By Lemma 3, 4 case (ii) will not hold. □

Lemma 7. *An evaluator P_i sends* HSig *to the garblers only if P_i has identified the co-evaluator to be honest.*

Proof. If P_i is corrupt and sends HSig, then her co-evaluator P_j is honest. If P_i is honest, then P_i sends HSig only if P_i is unable to evaluate the GC due to the violation Lemma 5 and $\mathsf{Corr}_i = \{P_k\}$ or $\mathsf{Con}_i = (P_k, P_l)$ where P_k, P_l are garblers. By Lemma 3, 4, both P_k, P_l cannot be honest. Hence P_j is honest and the lemma. □

Lemma 8. *An honest garbler accepts the encoded output $\mathbf{Y}$ only if circuit evaluation was performed on committed inputs.*

Proof. An honest garbler P_i accepts the encoded output only if (i). $\mathbf{Y}/\tilde{\mathbf{Y}}$ was received from an evaluator P_j not in Corr_i or in conflict with P_i's co-garbler. (ii). $\mathbf{Y}/\tilde{\mathbf{Y}}$ was received from P_j who was in conflict with P_i's co-garbler but received (HSig, P_j) from P_j's co-evaluator (honest). In case (i), by Lemma 5 the GC was evaluated on committed inputs by both evaluators and case (ii) follows from Lemma 7. □

Lemma 9. *If an honest evaluator P_3 receives no d from either garblers, but receives $(\tilde{\mathbf{Y}}, \tilde{o_4})$ from co-evaluator, then P_3 accepts it.*

Proof. An honest evaluator P_3 does not receive the decoding information from an honest garbler P_j only if P_3 was in conflict with P_j's co-garbler at the end of round 2. This further implies corruption to be among the garblers and thus P_3 accepts $(\tilde{\mathbf{Y}}, \tilde{o_4})$ sent by P_4 who is found to be honest by P_3. □

Lemma 10. *The protocol* 3RFair *is correct.*

Proof. We prove that the output y is computed on unique inputs committed in $\mathsf{InputCommit}$. By Lemma 1, a corrupt party is forced to commit to unique input while the honest parties' inputs are established by the end of round 1 of $\mathsf{InputCommit}$. If a corrupt party is not identified at the end of $\mathsf{InputCommit}$ and the encoded inputs received correspond to the committed input shares established at the end of round 2 of $\mathsf{InputCommit}$, by Lemma 5, an honest evaluator P_i computes the encoded output. Else if P_i does not evaluate and determined one of the garblers to be corrupt, by Lemma 7, P_i sends an HSig for garblers to accept the output only if her co-evaluator is honest. A corrupt evaluator also never obtains the decoding information for evaluation done on inputs that are not committed (by Lemma 6). Lastly, Lemmas 8 and 9 ensure that output if accepted, corresponds to the inputs committed at the end of $\mathsf{InputCommit}$. □

4.4.2 Security

This section presents the security proof of Theorem 1 which states the security of the protocol $\mathsf{3RFair}$ relative to its ideal functionality.

Proof. We describe the simulator $\mathcal{S}_{\mathsf{3RFair}}$ for the case when P_1 and P_3 are corrupt. The corruption of P_2 and P_4 are symmetric to the case when P_1 and P_3 are corrupt, respectively. We will first describe the simulation for the $\mathsf{InputCommit}_i()$ routine, specifically for $\mathsf{InputCommit}_1()$ ($\mathsf{InputCommit}_2()$, $\mathsf{InputCommit}_3()$ and $\mathsf{InputCommit}_4()$ are symmetric). We will consider, separately, the cases when P_1 and P_2 are corrupt in $\mathsf{InputCommit}_1()$. Corruption of P_3, P_4 in $\mathsf{InputCommit}_1()$ is symmetric to the case of P_2's corruption. Simulation of $\mathsf{InputCommit}_1$ for corrupt P_1 and P_2 is described in Figures 4.3, 4.4 respectively.

We now give a brief overview of the simulation for $\mathsf{3RFair}$. When P_1 is corrupt, the simulator, acting on behalf of the honest parties can extract the input (either the committed input sent to majority of the honest parties or a default value) x_1 of P_1 by the end of Round 1. Depending on how P_1 behaves in Round 2, the ideal functionality $\mathcal{F}_{\mathsf{fair}}$ is invoked by the simulator on adversary's behalf with either `abort` or x_1 to obtain the output. In case of simulation of P_3, as before, P_3 can decide to let the execution succeed or fail upto Round 2. This forces the simulator to invoke the ideal functionality based on P_3's behaviour only at the end of Round 2 and get the corresponding ouput. A technicality arises, since the simulator, on behalf on the honest parties, is required to send the GC, commitment on encoding and decoding information in Round 2 itself without knowing the output. To address this, it invokes the oblivious simulator of the GC, $\mathcal{S}_{\mathsf{obv}}$, which doesn't take the output and also doesn't return the decoding information. This results in the simulator committing to a dummy value for one share of the decoding information. Subsequently, when $\mathcal{F}_{\mathsf{fair}}$ is invoked to obtain y, the privacy simulator for the GC,

$\mathcal{S}_{\mathbf{priv}}$, is invoked with the same randomness and y, which returns the decoding information that makes the fake GC output y. The simulator then equivocates the commitment on the dummy value to the correct share of the decoding information. All of this pertains to the GC that P_3 evaluates. For the GC to be evaluated by P_4, P_3 receives the commitment to both shares of the decoding information and opening to only one of them. Similar equivocation steps are followed for the other commitment that is not opened, depending on P_3's behaviour. The details of the simulator when P_1 and P_3 are corrupt are described in Figures 4.5, 4.6.

$\underline{P_1^\star \text{ is corrupt}}$

Round 1:

- Receive commitments $(\mathsf{pp}_1, c_{12}, c_{13}, c_{14})$ on behalf of each of P_2, P_3 and P_4. Receive o_{12} on behalf of P_3, P_4; o_{13} on behalf of P_2, P_4; and o_{14} on behalf of P_2, P_3.
- For $P_k \in \mathbf{P}_1$, set $\mathsf{Corr}_k = \{P_1\}$ if any of the following is true:

a. $\mathsf{Open}(\mathsf{pp}_1, c_{1j}, o_{1j}) = \perp$ for any $j \in \mathsf{ind}(\mathbf{P}_{1k})$.

b. there does not exist a majority among the versions of $(\mathsf{pp}_1, c_{12}, c_{13}, c_{14})$ received on behalf of parties in $\mathbf{P}_1$. Assume a default value for P_1''s input share in this case. If there exists a majority, set c_{1j} as the majority value and o_{1j} as the corresponding opening. Compute $x_1 = x_{12} \oplus x_{13} \oplus x_{14}$ where $x_{1j} = \mathsf{Open}(\mathsf{pp}_1, c_{1j}, o_{1j})$ for $j \in \{2, 3, 4\}$.

c. for $j \in \mathsf{ind}(\mathbf{P}_1)$ if (pp_1, c_{1j}) is different from majority c_{1j}.

- For $P_k \in \mathbf{P}_1$, set $\mathsf{Con}_k = \mathsf{Con}_k \cup (P_1, P_l)$ where $l \in \mathsf{ind}(\mathbf{P}_{1k})$ if different copies of $(\mathsf{pp}_1, c_{12}, c_{13}, c_{14})$ are received on behalf of P_k and P_l.

Figure 4.3: Simulator $\mathcal{S}^1_{\mathsf{InputCommit}_1}$

$\underline{P_2^\star \text{ is corrupt}}$

Round 1:

- On behalf of P_1, sample pp_1, compute c_{1j} as commitment on randomly chosen x_{1j} for $j \in \mathsf{ind}(\mathbf{P}_{12})$ (input shares of P_1 available with $P_2^\star$), and as commitment to dummy values for $j \notin \mathbf{P}_{12}$. Send $(\mathsf{pp}_1, c_{12}, c_{13}, c_{14})$ and (o_{13}, o_{14}) to $P_2^\star$.

Round 2:

- Send $(\mathsf{pp}_1, c_{12}, c_{13}, c_{14})$ and o_{14} on behalf of P_3 to $P_2^\star$. Send $(\mathsf{pp}_1, c_{12}, c_{13}, c_{14})$ and o_{13} to $P_2^\star$ on behalf of P_4.
- Receive $(\mathsf{pp}_1', c_{12}', c_{13}', c_{14}')$ on behalf of both P_3 and P_4. Also, receive o_{13}' and o_{14}' on behalf of P_4 and P_3, respectively.
- For $P_k \in \mathbf{P}_{12}$ set $\mathsf{Con}_k = \mathsf{Con}_k \cup (P_1, P_2)$ if version of commitment received from $P_2^\star$ is not same as that sent by P_1 in Round 1.
- Set $\mathsf{Corr}_k = P_2$ if opening received on behalf of P_k is different from what was sent in Round 1.

Figure 4.4: Simulator $\mathcal{S}^2_{\mathsf{InputCommit}_1}$

$P_1^\star$ is corrupt

Round 1:

- Simulation of Round 1 of $\mathcal{S}_{\mathsf{InputCommit}_\alpha}$ for $\alpha \in [4]$ (refer figure 4.3, 4.4).
- Receive seed and share of public parameter $(r_3, \mathsf{epp}_1^3), (r_4, \mathsf{epp}_1^4)$ from $P_1^\star$ on behalf of P_2 (for $\mathsf{Garble}_{34}, \mathsf{Garble}_{43}$, respectively).
- Send $[\tilde{o}_{31}^b]_0, [\tilde{o}_{32}^b]_0$ for $b \in \{0,1\}$ on behalf of P_3, and $[o_{41}^b]_0, [o_{42}^b]_0$ for $b \in \{0,1\}$ on behalf of P_4 to $P_1^\star$.

Round 2:

- Simulation of Round 2 of $\mathcal{S}_{\mathsf{InputCommit}_\alpha}$ for $\alpha \in [4]$.
- On behalf of P_3: Receive $\mathbf{B}_3$ comprising of the GC to be evaluated by P_3, commitments on encoding information, commitment on shares of decoding information, opening of the first share of the decoding information and permutation strings p_{ij} for $i \in [4], j \in \mathsf{ind}(\mathbf{P}_{i3})$. Receive $\mathbf{B}_4'$ comprising of commitment on shares of decoding information for the GC to be evaluated by P_4 and opening of the first share of this decoding information. Receive openings corresponding to input labels for $x_{ij}, i \in [4], j \in \mathsf{ind}(\mathbf{P}_{i1})$ and masked input shares. Also, receive $\{[o_{41}^b]_1, [o_{42}^b]_1\}_{b\in\{0,1\}}$ for input wires belonging to shares x_{41}, x_{42}.
- Similarly, receive $\mathbf{B}_4$, $\mathbf{B}_3'$, and opening of respective labels on behalf of P_4.
- Set $\mathbf{conflict} = 1$ and $\mathsf{Con}_3 = \mathsf{Con}_3 \cup (P_1, P_2)$ if any of the following is true:

a. Hash of the received $\mathbf{B}_3$ or $\mathbf{B}_4'$ does not match the hash of $\mathbf{B}_3$ or $\mathbf{B}_4'$, respectively, computed using the received random seed r_3 and epp_3^1.

b. For shares not possessed by P_3, opening of encoding information received does not correspond to the originally committed label (known on behalf of P_2) or if the masked input shares mismatch.

Similar steps are carried out with respect to P_4.

- Set $\mathsf{Con}_3 = \mathsf{Con}_3 \cup (P_1, P_4)$ if opening of encoding information for x_{42} does not correspond to the originally committed label.
- Set $\mathbf{corrupt}_1 = 1$ and $\mathsf{Corr}_3 = P_1$ if for shares possessed by P_3 (excluding x_{42}), opening of encoding information received is different from the originally committed labels (known on behalf of P_2) or if there is a mismatch among the $\{m_{ij}\}_{i\in[4], j\in\mathsf{ind}(\mathbf{P}_{i3})}$ received from P_1 and the ones computed from x_{ij} and p_{ij} received from P_1. Similar steps are carried out with respect to P_4.
- If for, both, P_3, P_4, $\mathbf{corrupt}_1 = 1$, or $\mathbf{conflict} = 1$, or $\mathbf{corrupt}_1 = 1$ for one of them and $\mathbf{conflict} = 1$ for the other, then invoke $\mathcal{F}_{\mathsf{fair}}$ with $(\mathsf{Input}, \mathtt{abort})$ on behalf of $P_1^\star$. Else, using x_1 computed in $\mathcal{S}^1_{\mathsf{InputCommit}_1}$, invoke $\mathcal{F}_{\mathsf{fair}}$ with (Input, x_1) to obtain y.

Round 3:

- Receive o_4 and $\tilde{o_3}$ on behalf of P_3 and P_4 respectively.
- Compute $\mathbf{Y}$ on behalf of P_3 such that $\mathsf{De}(\mathbf{Y}, d) = y$ (d known on behalf of P_2). Similarly compute $\tilde{\mathbf{Y}}$ on behalf of P_4.

- If P_3 could not have computed $\mathbf{Y}$ and $P_4 \notin \mathsf{Con}_3 \cup \mathsf{Corr}_3$, then send (P_4, HSig) on behalf of P_3. Else if $y \neq \perp$ send $\mathbf{Y}$ to $P_1^\star$ on behalf of P_3. Similar steps are carried out by P_4 for $\tilde{\mathbf{Y}}$.

Figure 4.5: Simulator $\mathcal{S}^1_{\mathsf{3RFair}}$

Security against corrupt $P_1^\star$: We argue that $\text{IDEAL}_{\mathcal{F}_{\mathsf{fair}}, \mathcal{S}^1_{\mathsf{3RFair}}} \overset{c}{\approx} \text{REAL}_{\mathsf{3RFair}, \mathcal{A}}$ when $\mathcal{A}$ corrupts P_1 based on the following series of intermediate hybrids.

HYB_0: Same as $\text{REAL}_{\mathsf{3RFair}, \mathcal{A}}$.

HYB_1: Same as HYB_0 except that $\{c_{ij}\}_{i \in \mathsf{ind}(\mathbf{P}_1), j \notin \mathsf{ind}(\mathbf{P}_{i1})}$ is replaced with a commitment to a dummy value in $\mathsf{InputCommit}_i$ when P_1 doesn't get access to the corresponding opening information.

HYB_2: Same as HYB_1 except that P_1 is added to $\mathsf{Corr}_k, k \in \mathsf{ind}(\mathbf{P}_1)$ if opening forwarded from P_1 to P_k corresponding to P_i's committed share ($i \in \mathsf{ind}(\mathbf{P}_{1k})$) in $\mathsf{InputCommit}_i$ is different from what was originally committed.

HYB_3: Same as HYB_2 except that if Corr_k and Con_k ($k \in \{3, 4\}$) are empty at the end of Round 2, then add P_1 to Corr_k if P_k receives anything other than the encoding information corresponding to shares x_{ij} that P_k possesses.

HYB_4: Same as HYB_3 except that if Corr_k and Con_k ($k \in \{3, 4\}$) are empty at the end of Round 2, then add (P_1, P_2) to Con_k if P_k receives anything other than the encoding information corresponding to shares x_{ij} that P_k does not possess.

HYB_5: Same as HYB_4 except that $\mathbf{Y}$ is computed such that $\mathsf{De}(\mathbf{Y}, d) = y$ instead of $\mathbf{Y} = \mathsf{Ev}(\mathsf{GC}, \mathbf{X})$.

Since $\text{HYB}_5 := \text{IDEAL}_{\mathcal{F}_{\mathsf{fair}}, \mathcal{S}^1_{\mathsf{3RFair}}}$, to conclude the proof we show that every two consecutive hybrids are indistinguishable.

$\text{HYB}_0 \overset{c}{\approx} \text{HYB}_1$: The difference between the hybrids is that when P_1 does not get access to openings of $\{c_{ij}\}_{i \in \mathsf{ind}(\mathbf{P}_1), j \notin \mathsf{ind}(\mathbf{P}_{i1})}$, in HYB_0, c_{ij} is a commitment on x_{ij}, whereas in HYB_1, it is a commitment on a dummy value. Indistinguishability follows from the hiding of the commitment scheme.

$\text{HYB}_1 \overset{c}{\approx} \text{HYB}_2$: The difference between the hybrids is that in HYB_1, P_1 is added to Corr_k if the opening forwarded by P_1 to P_k in $\mathsf{InputCommit}_i$ $i \in \mathsf{ind}(\mathbf{P}_{1k})$ results in $\perp$, whereas in HYB_2, P_1 is added to Corr_k if the opening sent by P_1 is anything other than what was originally committed. Since the probability with which P_1 could successfully decommit to something other than what was originally committed is negligible (due to the binding of the commitment scheme), the hybrids are indistinguishable.

$\text{HYB}_2 \overset{c}{\approx} \text{HYB}_3$: The difference between the hybrids is that when Corr_k and Con_k are empty, in

HYB_2 P_1 is added to Corr_k if the decommitment to shares $x_{ij}, i \in [4], j \in \mathsf{ind}(\mathbf{P}_{ik})$ sent by P_1 result in a $\perp$, whereas in HYB_3, P_1 is added to Corr_k if the decommitments open to any value other than the originally committed encoding information. Since the probability with which P_1 could successfully decommit to something other than what was originally committed is negligible (due to the binding of the commitment scheme), the hybrids are indistinguishable.

$\text{HYB}_3 \overset{c}{\approx} \text{HYB}_4$: The difference between the hybrids is that when Corr_k and Con_k are empty, in HYB_3 P_1 is added to Con_k if the decommitment to shares $x_{ij}, i \in [4], j \notin \mathsf{ind}(\mathbf{P}_{ik})$ sent by P_1 is inconsistent with that known on behalf of P_2, whereas in HYB_4, P_1 is added to Con_k if the decommitments open to any value other than the originally committed encoding information. Since the probability with which P_1 could successfully decommit to something other than what was originally committed is negligible (due to the binding of the commitment scheme), the hybrids are indistinguishable.

$\text{HYB}_4 \overset{c}{\approx} \text{HYB}_5$: The difference between the hybrids is that in HYB_4 $\mathbf{Y}$ is computed as $\mathsf{Ev}(\mathsf{GC}, \mathbf{X})$ whereas in HYB_5, it is computed such that $\mathsf{De}(\mathbf{Y}, d) = y$. Due to the correctness of the garbling sheme, the equivalence of $\mathbf{Y}$ holds.

$\underline{P_3^\star \text{ is corrupt}}$

Round 1:

- Simulation of Round 1 of $\mathcal{S}^3_{\mathsf{InputCommit}_\alpha}, \alpha \in [4]$.
- Receive $[\tilde{o}^b_{31}]_0, [\tilde{o}^b_{32}]_0$ for $b \in \{0,1\}$ on behalf of P_1 and P_2 (for Garble_{43}).

Round 2:

- Simulation of Round 2 of $\mathcal{S}^3_{\mathsf{InputCommit}_\alpha}, \alpha \in [4]$.
- Use uniform randomness r on behalf of P_1, P_2 and run $(\mathsf{GC}, \mathbf{X}) \leftarrow \mathcal{S}_{\mathsf{obv}}(1^\kappa, C)$ where $\mathcal{S}_{\mathsf{obv}}$ is the oblivious simulator of the garbling scheme. Choose $\{m_{ij}\}_{i\in[4], j\in\mathsf{ind}(\mathbf{P}_i)}$ at random. Let $\{c^b_{ij}\}_{i\in[4],j\in\mathsf{ind}(\mathbf{P}_i),b\in\{0,1\}}$ be commitments to entries of $\mathbf{X}$. Let $\mathbf{B}_3 = \{\mathsf{GC}, \{c^b_{ij}\}, c_3, c_4, o_3, \mathbf{p}_3\}$ where $p_{ij} \in \mathbf{p}_3$ (for $i \in [4], j \in \mathsf{ind}(\mathbf{P}_{i3})$), c_3, c_4, o_3 are computed as follows.
  - for $i \in \mathsf{ind}(\mathbf{P}_3), j \in \mathsf{ind}(\mathbf{P}_{i3})$, set $p_{ij} = x_{ij} \oplus m_{ij}$ consistent with the shares given to $P_3^\star$ during $\mathcal{S}^3_{\mathsf{InputCommit}_i}$.
  - for $i = 3, j \in \mathsf{ind}(\mathbf{P}_{33})$, compute p_{ij} with respect to the opening received on behalf of P_1, P_2 in $\mathcal{S}^3_{\mathsf{InputCommit}_3}$.
  - c_3 is computed as a commitment to a random share d_3 of the decoding information for the GC to be evaluated by $P_3^\star$, and o_3 is its corresponding opening. c_4 is generated using an equivocal commitment scheme (sample epp, with trapdoor t_1, t_2) as a commitment to a dummy value.

  Let $\mathbf{B}'_4 = \{\tilde{c}_3, \tilde{c}_4, \tilde{o}_3\}$ where $\tilde{c}_3$ is computed as a commitment to a random share $\tilde{d}_3$ (for P_4's GC), and $\tilde{o}_3$ is its corresponding commitment. $\tilde{c}_4$ is generated (using an equivocal commitment scheme) as a commitment to a dummy value.

- Send $\mathbf{B}_3, \mathbf{B}'_4$ and $\mathsf{H}(\mathbf{B}_3), \mathsf{H}(\mathbf{B}'_4)$ on behalf of P_1 and P_2 respectively.
- Send random $[o_{41}^{x_{41}}]_0, [o_{42}^{x_{42}}]_0$ on behalf of P_4.
- Send openings for input wires corresponding to the input shares $\{x_{ij}\}_{i\in[4],j\mathsf{ind}(\mathbf{P}_{ig})}$ for $g \in \{1,2\}$ and $\{[o_{41}^b]_1, [o_{42}^b]_1\}_{b\in\{0,1\}}$ on behalf of the garblers (Only P_1 sends opening corresponding to x_{34}) if corrupt set of the garbler is empty. Also send the masked input shares.
- Receive $[\tilde{o}_{31}^{x_{31}}]_0, [\tilde{o}_{32}^{x_{32}}]_0$ on behalf of P_4.
- **Invoking $\mathcal{F}_{\mathsf{fair}}$:** If either of the following conditions are satisfied ($\mathbf{corrupt}_3 = 1$), invoke $\mathcal{F}_{\mathsf{fair}}$ with $(\mathsf{Input}, \mathtt{abort})$ and set $y = \perp$.

  Condition 1: $P_3^\star$ gave a minority commitment and opening on x_{32} to P_1 (during $\mathsf{InputCommit}_3$) and gave incorrect $[\tilde{o}_{32}^{x_{32}}]_0$ to P_4.

  Condition 2 $P_3^\star$ gave a minority commitment and opening on x_{31} to P_2 (during $\mathsf{InputCommit}_3$) and gave incorrect $[\tilde{o}_{31}^{x_{31}}]_0$ to P_4.

  Else, using x_3 computed in $\mathcal{S}^3_{\mathsf{InputCommit}_3}$, invoke $\mathcal{F}_{\mathsf{fair}}$ with (Input, x_3) to get y.
- **Invoking $\mathcal{S}_{\mathbf{priv}}$:** If $P_3^\star$ did not result in an abort ($\mathbf{corrupt}_3 = 0$), run $(\mathsf{GC}, \mathbf{X}, d) \leftarrow \mathcal{S}_{\mathbf{priv}}(1^\kappa, C, y)$ where $\mathcal{S}_{\mathbf{priv}}$ is the privacy simulator of the garbling scheme. Compute $\mathbf{Y}$ such that $\mathsf{De}(\mathbf{Y}, d) = y$. Let $d_4 = d \oplus d_3$. Generate $\tilde{\mathsf{GC}}$ to be evaluated by P_4 honestly. Compute $\tilde{\mathbf{Y}}$ such that $\mathsf{De}(\tilde{\mathbf{Y}}, \tilde{d}) = y$. Let $\tilde{d}_4 = \tilde{d} \oplus \tilde{d}_3$.

Round 3:

- Receive $\mathbf{Y} = \mathsf{Ev}(\mathsf{GC}, \mathbf{X})$ or $\mathsf{HSig}(P_4)$ on behalf of P_1, P_2. Receive $(\mathbf{Y}, o_3)$ and/or $\tilde{o}_3$ on behalf of P_4.
- If $y \neq \perp$, send $(\tilde{\mathbf{Y}}, \tilde{o}_4)$ where $\tilde{o}_4 = \mathsf{Equiv}(\tilde{c}_4, \tilde{d}_4, t_1, t_2)$.
- Send o_4 on behalf of the garblers if $P_3^\star$ is not in their corrupt set or in confict with the co-garbler, where $o_4 = \mathsf{Equiv}(c_4, d_4, t_1, t_2)$. Similarly, send o_4 (equivocated) on behalf of P_4 if $P_3^\star$ is not in its corrupt or conflict set.

Figure 4.6: Simulator $\mathcal{S}^3_{\mathsf{3RFair}}$

Security against corrupt $P_3^\star$: We argue that $\text{IDEAL}_{\mathcal{F}_{\mathsf{fair}},\mathcal{S}^3_{\mathsf{3RFair}}} \overset{c}{\approx} \text{REAL}_{\mathsf{3RFair},\mathcal{A}}$ when $\mathcal{A}$ corrupts P_3 based on the following series of intermediate hybrids.

HYB_0: Same as $\text{REAL}_{\Pi,\mathcal{A}}$.

HYB_1: Same as HYB_0 except that when P_3 doesn't get access to the opening of $\{c_{ij}\}_{i\in\mathsf{ind}(\mathbf{P}_3), j\notin\mathsf{ind}(\mathbf{P}_{i3})}$, sent by P_i in $\mathsf{InputCommit}_i$, the commitment is replaced with a commitment to a dummy value.

HYB_2: Same as HYB_1 except that P_3 is added to $\mathsf{Corr}_k, k \in \mathsf{ind}(\mathbf{P}_3)$ if opening forwarded from P_3 to P_k corresponding to P_i's committed share ($i \in \mathsf{ind}(\mathbf{P}_{3k})$) in $\mathsf{InputCommit}_i$ is different from what was originally committed.

HYB_3: Same as HYB_2, except that P_1, P_2 use uniform randomness instead of pseudo-

randomness.

HYB_4: Same as HYB_3 except the following:

$\text{HYB}_{4.1}$ When the execution results in P_3 getting access to labels corresponding to its non-committed input, the GC to be evaluated by P_3 is created as $(\mathsf{GC}, \mathbf{X}) \leftarrow \mathcal{S}_{\mathsf{obv}}(1^\kappa, C)$ and the commitment to unopened share (d_4) of the decoding information is created on a dummy value. Similarly, the unopened share $\tilde{d}_4$ of decoding information for $\tilde{\mathsf{GC}}$ (to be evaluated by P_4) is created on a dummy value.

$\text{HYB}_{4.2}$ When the execution results in P_3 getting access to labels corresponding to its committed input, the GC for P_3 is created as $(\mathsf{GC}, \mathbf{X}, d) \leftarrow \mathcal{S}_{\mathbf{priv}}(1^\kappa, C, y)$, the commitment to unopened share (d_4) of the decoding information is computed on $d_4 = d \oplus d_3$. Similarly, the unopened share $\tilde{d}_4$ of decoding information for $\tilde{\mathsf{GC}}$ (to be evaluated by P_4) is created honestly.

HYB_5: Same as HYB_4 except that $\tilde{\mathbf{Y}}$ is computed such that $\mathsf{De}(\tilde{\mathbf{Y}}, \tilde{d}) = y$ instead of $\tilde{\mathbf{Y}} = \mathsf{Ev}(\tilde{\mathsf{GC}}, \tilde{\mathbf{X}})$.

HYB_6: Same as HYB_5 except that the protocol results in `abort` if neither P_1 nor P_2 nor P_4 receive $\mathbf{Y}$, $\mathbf{Y}$, $(\mathbf{Y}, o_3)$, where $\mathbf{Y}$ is obtained from GC evaluation by P_3.

Since $\text{HYB}_6 := \text{IDEAL}_{\mathcal{F}, \mathcal{S}^3_{\mathsf{3RFair}}}$, to conclude the proof we show that every two consecutive hybrids are indistinguishable.

$\text{HYB}_0 \overset{c}{\approx} \text{HYB}_1$: The difference between the hybrids is that when the execution results in P_3 not getting access to openings of $\{c_{ij}\}_{i \in \mathsf{ind}(\mathbf{P}_3), j \notin \mathsf{ind}(\mathbf{P}_{i3})}$, in HYB_0, c_{ij} is a commitment on x_{ij}, while in HYB_1, it is a commitment on a dummy value. Indistinguishability follows from the hiding of the commitment scheme.

$\text{HYB}_1 \overset{c}{\approx} \text{HYB}_2$: The difference between the hybrids is that in HYB_1 P_3 is added to Corr_k if the opening forwarded by P_3 to P_k in $\mathsf{InputCommit}_i$ $i \in \mathsf{ind}(\mathbf{P}_{3k})$ results in $\perp$, whereas in HYB_2, P_3 is added to Corr_k if the opening sent by P_3 is anything other than what was originally committed. Since the probability with which P_3 could successfully decommit to something other than what was originally committed is negligible (due to the binding of the commitment scheme), the hybrids are indistinguishable.

$\text{HYB}_2 \overset{c}{\approx} \text{HYB}_3$: The difference between the hybrids is that in HYB_2 P_1, P_2 use pseudorandomness whereas in HYB_3 they use uniform randomness. Indistinguishability of the hybrids follows from the reduction to the security of PRG G.

$\text{HYB}_3 \overset{c}{\approx} \text{HYB}_{4.1}$: The difference between the hybrids is in the way $(\mathsf{GC}, \mathbf{X})$ is generated when the execution leads to P_3 getting access to labels corresponding to its non-committed input. In HYB_3, it is obtained as $(\mathsf{GC}, e, d) \leftarrow \mathsf{Gb}(1^\kappa, C)$ whereas in $\text{HYB}_{4.1}$ it is computed as

$(\mathsf{GC}, \mathbf{X}) \leftarrow \mathcal{S}_{\mathsf{obv}}(1^\kappa, C)$. Also, the commitment to the unopened share of decoding information is created on a dummy value. Indistinguishability of the hybrids follows from the reduction to the obliviousness of the garbling scheme and the hiding property of the commitment scheme.
$\text{HYB}_3 \overset{c}{\approx} \text{HYB}_{4.2}$: The difference between the hybrids is in the way $(\mathsf{GC}, \mathbf{X})$ is generated. In HYB_3, $(\mathsf{GC}, e, d) \leftarrow \mathsf{Gb}(1^\kappa, C)$ is run, whereas in $\text{HYB}_{4.2}$, it is generated as $(\mathsf{GC}, \mathbf{X}, d) \leftarrow \mathcal{S}_{\mathbf{priv}}(1^\kappa, C, y)$. Indistinguishability follows via reduction to the privacy of the garbling scheme and hiding of the commitment scheme.
$\text{HYB}_4 \overset{c}{\approx} \text{HYB}_5$: The difference between the hybrids is that in HYB_4 $\tilde{\mathbf{Y}}$ is computed as $\mathsf{Ev}(\tilde{\mathsf{GC}}, \tilde{\mathbf{X}})$ whereas in HYB_5, it is computed such that $\mathsf{De}(\tilde{\mathbf{Y}}, \tilde{d}) = y$. Due to the correctness of the garbling sheme, the equivalence of $\tilde{\mathbf{Y}}$ holds.
$\text{HYB}_5 \overset{c}{\approx} \text{HYB}_6$: The difference in the hybrids is that in HYB_5 the protocol aborts if neither P_1 nor P_2 nor P_4 receive $\mathbf{Y}$ such that $\mathsf{De}(\mathbf{Y}, d) \neq \perp$ from P_3 whereas in HYB_6, the protocol aborts if neither P_1nor P_2 nor P_4 receive $\mathbf{Y}$ obtained from GC evaluation. The probability that P_3 could have sent a $\mathbf{Y}$ such that $\mathbf{Y} \neq \mathsf{Ev}(\mathsf{GC}, \mathbf{X})$ but $\mathsf{De}(\mathbf{Y}, d) \neq \perp$ is negligible due to the authenticity of the garbling scheme. Thus, the hybrids are indistinguishable.

□

Chapter 5

GOD in 3 Rounds

Having achieved fairness in 3 rounds at the expense of 2 GCs in 3RFair, we now aim to attain GOD in 3 rounds that involves communication and computation of 4 GCs in the worst case.

5.1 The Construction

We begin with 3RFair and tackle all the abort cases to attain GOD. It is easy to see that, when the adversary does not misbehave until round 2, our 3RFair substitutes for GOD. We observe that, in most cases, when the misbehavior detected within two rounds, leads to abort in round 3 of 3RFair (due to violation of the invariant of evaluation on committed inputs), the adversary exposes herself as corrupt to two honest parties (say P_i, P_j). Further, P_i, P_j know each other's identity and thus can exchange their committed input-shares in round 3 to compute the output. RSS ensures that the shares of two parties are sufficient to know all inputs. P_i, P_j reveal their shares to the remaining honest party P_h for her to compute the output. However, to enable the corrupt party to compute the output (as shares cannot be revealed), P_i, P_j construct an additional GC which takes RSS shares owned by P_i, P_j as inputs. This GC is constructed with shared randomness sampled in one of the first two rounds and one of P_i, P_j sends the GC to the corrupt party while both send encoded labels wrt their RSS shares in round 3. We also give away decoding information for this extra GC to allow the corrupt party to obtain the output on evaluation. This process causes an overhead of 1 GC compared to 3RFair and suffices for all but below case.

A potential corrupt garbler may send incorrect copy of GC to both the evaluators in round 2. In such case, the evaluators can best identify conflict among the garblers, one of whom is corrupt. As a result, the evaluators generate two additional augmented GCs as in the previous case, one for each garbler and send the GCs along with the encoded labels for their RSS shares

to each garbler individually to help them compute the output. In this case alone, the overhead amounts to 2 GCs over $\mathsf{3RFair}$ leading to an overall use of 4 GCs for GOD and accounts for the worst case execution of our protocol. In each of the above cases, an honest party prepares the additional GC with appropriate randomness for the corrupt party/conflict parties at the end of round 2 if it anticipates an impending abort in round 3 and communicates the same in round 3. The formal description of augmented circuit appears in Figs 5.1. The formal protocol which uses a garble routine (Fig 5.2) to construct the additional GC (in case of impending abort in $\mathsf{3RFair}$) is presented in Fig 5.3.

Input: P_i has input x_i, x_{mn} for $m \in \mathsf{ind}(\mathbf{P}_i), n \in \mathsf{ind}(\mathbf{P}_{im})$. P_j has input x_j, x_{mn} for $m \in \mathsf{ind}(\mathbf{P}_j), n \in \mathsf{ind}(\mathbf{P}_{jm})$.

Output: $y = f(x_1, x_2, x_3, x_4)$.

Computation: The circuit computes output as:
For $k \in \mathsf{ind}(\mathbf{P}_{ij})$, compute $x_k = \oplus_{\ell \in \mathsf{ind}(\mathbf{P}_k)} x_{k\ell}$ using $x_{k\ell}$ given by P_i and P_j as input. Output $y = f(x_1, x_2, x_3, x_4)$.

Figure 5.1: Circuit Description Ckt

Input: Let $\{i, j\} = \mathsf{ind}(\mathbf{P}_{kl})$. P_i has input x_i, x_{mn} for $m \in \mathsf{ind}(\mathbf{P}_i), n \in \mathsf{ind}(\mathbf{P}_{im})$. P_j has input x_j, x_{mn} for $m \in \mathsf{ind}(\mathbf{P}_j), n \in \mathsf{ind}(\mathbf{P}_{jm})$.

Output: P_k outputs $y = f(x_1, x_2, x_3, x_4)$.

Common Input: The circuit Ckt (as described in Fig 5.1).

Round 1:
- P_i and P_j use r_{ij} to garble the circuit Ckt (Fig 5.1) as $(\mathsf{GC}, e, d) \leftarrow \mathsf{Gb}(1^\kappa, \mathsf{Ckt})$. P_i sends (GC, d) to P_k.
- Assume $\{e_w^0, e_w^1\}$ be the encoding information for wire w. P_i sends encoding labels for each input wire w corresponding to value α in V_i i.e. e_w^α to P_k. Similarly P_j sends encoded labels for input wire w corresponding to value β in V_j i.e. e_w^β to P_k.

Local Computation: Set $\mathbf{X} = \{e_w\}_{w \in I}$, I denotes the set of all input wires. P_k computes $\mathbf{Y} = \mathsf{Ev}(\mathsf{GC}, \mathbf{X})$ and $y = \mathsf{De}(\mathbf{Y}, d)$. Output y.

Figure 5.2: Abort GC routine GC_k^ℓ

Input and Output: P_i inputs x_i and outputs $y = f(x_1, x_2, x_3, x_4)$.

Round 1: Run round 1 of 3RFair (Fig 4.2). Besides,

- For each $i, j \in [4]$ where $i < j$, P_i samples a random PRF seed $r_{ij} \in \{0,1\}^\kappa$ and sends to P_j.
- Initialize flags $\mathbf{corrupt}_i = 0$ and $\mathbf{honest}_i = 0$ for $i \in [4]$.

Round 2:

- Run round 2 of 3RFair (Fig 4.2) except that strong NICOM is used to commit on shares of decoding information instead of eNICOM.
- Set flags $\mathbf{corrupt}_i$ and $\mathbf{honest}_i$ as per **blue** highlighted text in 3RFair.

Round 3: Run round 3 of 3RFair (Fig 4.2). Additionally,

- If there exist k, ℓ s.t $\mathbf{corrupt}_k = 1$ and $\mathbf{honest}_\ell = 1$, do the following:
  - Let $i, j = \mathsf{ind}(\mathbf{P}_{k\ell})$ such that $i < j$. P_i gives V_i to P_j and P_ℓ. P_j gives V_j to P_i and P_ℓ.
  - P_i, P_j, P_ℓ compute y from V_i and V_j. Output y.
  - Run routine GC_k^ℓ (Fig 5.2).
- Else, if $\mathbf{conflict} = 1$, run routine instances GC_1^2 and GC_2^1 (Fig 5.2).

Figure 5.3: 3-round GOD protocol 3RGod

The optimizations accounted for 3RFair can also be utilized in our 3RGod protocol. The sending of additional GCs can also be optimized similar to the GCs sent in round 2.

5.2 Correctness and Security

Theorem 2. *Assuming one-way permutations exist, the protocol* 3RGod *(Fig 5.3) securely realizes the functionality* $\mathcal{F}_{\mathsf{god}}$ *(Fig 2.1) tolerating one malicious corruption in the standard model.*

5.2.1 Correctness

Lemma 11. *The protocol* 3RGod *is correct.*

Proof. We prove that the output y computed corresponds to unique inputs committed in InputCommit. A corrupt party is forced to commit to her input or a default value is chosen. The correctness of the protocol for all but abort cases follows from the correctness of 3RFair (Lemma 10). For the abort cases, the additional GCs are prepared by honest parties and hence are correct. Since the inputs are committed by the end of round 2 of InputCommit, the views exchanged by honest parties and the encoded input sent by honest parties to the

additional GC evaluator in round 3, correspond to the committed inputs. Thus the output y that is computed always corresponds to the uniquely committed inputs. □

5.2.2 Security

This section presents the security proof of Theorem 2 which states the security of the protocol 3RGod relative to its ideal functionality.

Proof. We describe the simulator $\mathcal{S}_{\mathsf{3RGod}}$ for the case when P_1 and P_3 are corrupt. The corruption of P_2 and P_4 are symmetric to the case when P_1 and P_3 are corrupt, respectively. The simulation for 3RGod is similar to that of 3RFair. When P_1 is corrupt, the simulator acting on behalf of the honest parties can extract x_1 at the end of Round 1 itself. Thus, the simulator can invoke the ideal functionality $\mathcal{F}_{\mathsf{god}}$ with x_1 on behalf of the adversary at the end of Round 1 to obtain the output y. However, for the case when P_3 is corrupt, either the oblivious simulator or the privacy simulator has to be invoked depending on whether P_3 gets access to input labels corresponding to non-committed input shares or not, respectively, in Round 2. Since the simulator, acting on behalf of the honest parties, knows this at the end of Round 1 itself, it can invoke the oblivious simulator when P_3 doesn't get access to labels on its committed shares. In this case, the commitment on the second share of the decoding information is done on a dummy value. When P_3 behaves honestly, the privacy simulator is invoked and the commitments to shares of decoding information are generated honestly. With these details we describe the simulators in Figures 5.4, 5.5. While describing the simulator, we only give the extra steps carried out in addition to steps involved in $\mathcal{S}_{\mathsf{3RFair}}$. Please refer to Figures 4.5, 4.6 for $\mathcal{S}_{\mathsf{3RFair}}$.

$P_1^\star$ is corrupt

Round 1:

- Simulation of Round 1 of $\mathcal{S}^1_{\mathsf{3RFair}}$.
- Receive seeds r_{12}, r_{13}, r_{14} on behalf of P_2, P_3 and P_4, respectively.
- Use the x_1 extracted by the simulator in the simulation of $\mathsf{InputCommit}_1$ to invoke the $\mathcal{F}_{\mathsf{god}}$ functionality with (Input, x_1) to obtain output y.

Round 2:

- Simulation of Round 2 of $\mathcal{S}^1_{\mathsf{3RFair}}$ up and until the population of conflict and corrupt sets.
- If garblers are found to be in conflict, set $\mathbf{conflict} = 1$. If a corrupt garbler P_g is identified, set $\mathbf{corrupt}_g = 1$ and $\mathbf{honest}_j = 1$ where $g \in \{1, 2\}$ and $j \in [2] \setminus \{g\}$ (as in Round 2 of $\mathcal{S}^1_{\mathsf{3RFair}}$).

Round 3:

- Simulation of Round 3 of $\mathcal{S}^1_{\mathsf{3RFair}}$ with the exception that instead of checking whether $y \neq \perp$ check if for, both, P_3, P_4, $\mathbf{corrupt}_1 = 1$, or $\mathbf{conflict} = 1$, or $\mathbf{corrupt}_1 = 1$ for one of them and $\mathbf{conflict} = 1$ for the other. If the condition is false, then continue as in Round 3 of $\mathcal{S}^1_{\mathsf{3RFair}}$.
- Else, if the above condition is true, generate $(\mathsf{GC}, e, d) \leftarrow \mathcal{S}_{\mathbf{priv}}(1^\kappa, \mathsf{Ckt}, y)$. Send (GC, d), labels for each wire corresponding to values in V_3 on behalf of P_3. Send labels corresponding to values in V_4 on behalf of P_4.

Figure 5.4: Simulator $\mathcal{S}^1_{\mathsf{3RGod}}$

Security against corrupt $P_1^\star$: We argue that $\text{IDEAL}_{\mathcal{F}_{\mathsf{god}}, \mathcal{S}^1_{\mathsf{3RGod}}} \overset{c}{\approx} \text{REAL}_{\mathsf{3RGod}, \mathcal{A}}$ when $\mathcal{A}$ corrupts P_1 based on the following series of intermediate hybrids.

HYB_0: Same as $\text{REAL}_{\mathsf{3RGod}, \mathcal{A}}$.

HYB_1: Same as HYB_0 except that we run steps from $\mathcal{S}^1_{\mathsf{3RFair}}$.

HYB_2: Same as HYB_1 except that when P_1 gets output from the GC generated by the parties who identified P_1 to be corrupt, the GC is generated as $(\mathsf{GC}, e, d) \leftarrow \mathcal{S}_{\mathbf{priv}}(1^\kappa, \mathsf{Ckt}, y)$.

Since $\text{HYB}_2 := \text{IDEAL}_{\mathcal{F}_{\mathsf{god}}, \mathcal{S}^1_{\mathsf{3RGod}}}$, to conclude the proof we show that every two consecutive hybrids are indistinguishable.

$\text{HYB}_0 \overset{c}{\approx} \text{HYB}_1$: The difference between the hybrids is that in HYB_0 steps from $\mathsf{3RFair}$ are run, whereas in HYB_1 steps from $\mathcal{S}^1_{\mathsf{3RFair}}$ are run. The hybrids are indistinguishable owing to the indistinguishability of $\text{IDEAL}_{\mathcal{F}_{\mathsf{fair}}, \mathcal{S}^1_{\mathsf{3RFair}}}$ and $\text{REAL}_{\mathsf{3RFair}, \mathcal{A}}$.

$\text{HYB}_1 \overset{c}{\approx} \text{HYB}_2$: The difference between the hybrids is in the way (GC, e, d) is generated. In HYB_1, $(\mathsf{GC}, e, d) \leftarrow \mathsf{Gb}(1^\kappa, \mathsf{Ckt})$ is run, whereas in HYB_2, it is generated as $(\mathsf{GC}, e, d) \leftarrow \mathcal{S}_{\mathbf{priv}}(1^\kappa, \mathsf{Ckt}, y)$. Indistinguishability follows via reduction to the privacy of the garbling scheme.

$\underline{P_3^\star \text{ is corrupt}}$

Round 1:

- Simulation of Round 1 of $\mathcal{S}^3_{\mathsf{3RFair}}$.
- Receive r_{34} on behalf of P_4.
- Use x_3 extracted by the simulator in the simulation of $\mathsf{InputCommit}_3$ to invoke $\mathcal{F}_{\mathsf{god}}$ functionality wit (Input, x_3) to obtain output y.

Round 2:

- Run Round 2 of $\mathcal{S}^3_{\mathsf{3RFair}}$ upto the point before the ideal functionality is invoked with the following exception.
  a. If $P_3 \notin \{\mathsf{Corr}_i\}_{i \in \{1,2\}}$ and $(P_1, P_3) \notin \mathsf{Con}_2$ and $(P_2, P_3) \notin \mathsf{Con}_1$, then run $(\mathsf{GC}, e, d) \leftarrow \mathcal{S}_{\mathbf{priv}}(1^\kappa, C, y)$ instead of $(\mathsf{GC}, \mathbf{X}) \leftarrow \mathcal{S}_{\mathsf{obv}}(1^\kappa, C)$. In this case, let $d = d_3 \oplus d_4$. Commitments c_3, c_4 are generated on d_3 and d_4. Else run Round 2 of $\mathcal{S}^3_{\mathsf{3RFair}}$ as such.

b. Generate the $\tilde{\mathsf{GC}}$ for P_4 honestly and let $\tilde{d} = \tilde{d}_3 \oplus \tilde{d}_4$. Generate $\tilde{B}'_4 = \{\tilde{c}_3, \tilde{c}_4, \tilde{o}_3\}$ where $\tilde{c}_3$ is computed as commitment to $\tilde{d}_3$ and $\tilde{o}_3$ is its corresponding opening. Similarly, $\tilde{c}_4$ is commitment to $\tilde{d}_4$ and $\tilde{o}_4$ is its corresponding opening.

Round 3:

- If $\mathbf{corrupt}_3 = 1$ and $\mathbf{honest}_l = 1$, generate $(\mathsf{GC}, e, d) \leftarrow \mathcal{S}_{\mathbf{priv}}(1^\kappa, \mathsf{Ckt}, y)$. Send (GC, d), labels for each wire corresponding to values in V_i on behalf of P_i and send labels corresponding to values in V_j on behalf of P_j where $i, j \in \mathsf{ind}(\mathbf{P}_{3l}), i < j$.
- Else, run Round 3 of $\mathcal{S}^3_{\mathsf{3RFair}}$ with the following exception. P_4 sends $(\tilde{\mathbf{Y}}, \tilde{o}_4)$ where $\tilde{o}_4$ is the honestly generated opening to $\tilde{d}_4$ instead of being equivocated, and $\tilde{\mathbf{Y}}$ is computed such that $\mathsf{De}(\tilde{\mathbf{Y}}, \tilde{d}) = y$. Similarly, o_4, if and when sent by P_1, P_2 is the honestly generated opening for d_4.

Figure 5.5: Simulator $\mathcal{S}^3_{\mathsf{3RGod}}$

Security against corrupt $P_3^\star$: We argue that $\text{IDEAL}_{\mathcal{F}_{\mathsf{god}}, \mathcal{S}^3_{\mathsf{3RGod}}} \overset{c}{\approx} \text{REAL}_{\mathsf{3RGod}, \mathcal{A}}$ when $\mathcal{A}$ corrupts P_3 based on the following series of intermediate hybrids.

HYB_0: Same as $\text{REAL}_{\Pi, \mathcal{A}}$.

HYB_1: Same as HYB_0 except that we run steps from $\mathcal{S}^3_{\mathsf{3RFair}}$.

HYB_2: Same as HYB_1 except that when P_3 gets output from the GC generated by the parties who identified P_3 to be corrupt, the GC is generated as $(\mathsf{GC}, e, d) \leftarrow \mathcal{S}_{\mathbf{priv}}(1^\kappa, \mathsf{Ckt}, y)$.

Since $\text{HYB}_2 := \text{IDEAL}_{\mathcal{F}, \mathcal{S}^3_{\mathsf{3RGod}}}$, to conclude the proof we show that every two consecutive hybrids are indistinguishable.

$\text{HYB}_0 \overset{c}{\approx} \text{HYB}_1$: The difference between the hybrids is that in HYB_0 steps from $\mathsf{3RFair}$ are run, whereas in HYB_1 steps from $\mathcal{S}^3_{\mathsf{3RFair}}$ are run. The hybrids are indistinguishable owing to the indistinguishability of $\text{IDEAL}_{\mathcal{F}_{\mathsf{fair}}, \mathcal{S}^3_{\mathsf{3RFair}}}$ and $\text{REAL}_{\mathsf{3RFair}, \mathcal{A}}$.

$\text{HYB}_1 \overset{c}{\approx} \text{HYB}_2$: The difference between the hybrids is in the way (GC, e, d) is generated. In HYB_1, $(\mathsf{GC}, e, d) \leftarrow \mathsf{Gb}(1^\kappa, \mathsf{Ckt})$ is run, whereas in HYB_2, it is generated as $(\mathsf{GC}, e, d) \leftarrow \mathcal{S}_{\mathbf{priv}}(1^\kappa, \mathsf{Ckt}, y)$. Indistinguishability follows via reduction to the privacy of the garbling scheme.

□

Chapter 6

GOD in 2 Rounds

In this section, we present a 2-round 4PC protocol achieving GOD. Towards achieving the goal of all parties obtaining the output in 2 rounds, we first observe that the latest a party can obtain the output is as an evaluator at the end of 2 rounds, in the GC based approach. Accordingly, our 2-round robust 4PC involves 4 executions, where each party enacts as an evaluator once and each execution comprises of 3 garblers. Due to the involved nature of the protocol, we first describe the 2-round protocol $\mathsf{2RGodSetup}()$ (Section 6.1) assuming the presence of a mild one-time setup: a common randomness has been shared amongst every set of 3 parties. This simplifies the description of the execution where these 3 parties act as garblers. The outline of this chapter is as follows: We first present the 2-round protocol $\mathsf{2RGodSetup}()$ involving 8 GCs and present a neat optimization that reduces its cost to 6 GCs. Next, with $\mathsf{2RGodSetup}()$ (non-optimized variant) as the stepping stone, we show how to remove the setup assumption in order to obtain our 2-round 4PC protocol $\mathsf{2RGod}()$ (Section 6.3) achieving GOD. We point that both our protocols are round-optimal and involve communication and computation of 6 GCs and 8 GCs respectively which is a considerable improvement over the 2-round protocol of [IKKP15] involving 12 GCs.

6.1 With one-time setup

At a high-level, our 2-round robust 4PC assuming setup $\mathsf{2RGodSetup}()$ involves parallel executions of sGod_i, $i \in [4]$ with P_i as evaluator and $\{P_j\}_{j \in \mathsf{ind}(\mathbf{P}_i)}$ as garblers. We describe a single execution $\mathsf{sGod}_4()$ (formally presented in Fig 6.2) with $\{P_1, P_2, P_3\}$ as garblers where P_4 as evaluator obtains the output at the end of 2 rounds.

Building upon the ideas of our $\mathsf{3RGod}$, we note that if P_4 must obtain the output on the *committed* (Definition 5) inputs of the parties at the end of Round 2, we need a way to establish

the committed inputs without Round 2 of $\mathsf{InputCommit}_i$ ($i \in [4]$). This is done by augmenting the circuit to incorporate the logic of Round 2 of $\mathsf{InputCommit}_i$. To elaborate, the circuit in sGod_4 takes as input the views of garblers $\{P_1, P_2, P_3\}$ i.e $\{\mathsf{V}_1, \mathsf{V}_2, \mathsf{V}_3\}$ after Round 1 of $\mathsf{InputCommit}_i(i \in [4])$ (described in Table 6.1).

Table 6.1: Table representing the views of all parties.

V_1	$\{c_{12}, c_{13}, c_{14}, o_{12}, o_{13}, o_{14}\}$, $\{c_{21}, c_{23}, c_{24}, o_{23}, o_{24}\}$ $\{c_{31}, c_{32}, c_{34}, o_{32}, o_{34}\}$, $\{c_{41}, c_{42}, c_{43}, o_{42}, o_{43}\}$
V_2	$\{c_{12}, c_{13}, c_{14}, o_{13}, o_{14}\}$, $\{c_{21}, c_{23}, c_{24}, o_{21}, o_{23}, o_{24}\}$ $\{c_{31}, c_{32}, c_{34}, o_{31}, o_{34}\}$, $\{c_{41}, c_{42}, c_{43}, o_{41}, o_{43}\}$
V_3	$\{c_{12}, c_{13}, c_{14}, o_{12}, o_{14}\}$, $\{c_{21}, c_{23}, c_{24}, o_{21}, o_{24}\}$ $\{c_{31}, c_{32}, c_{34}, o_{31}, o_{32}, o_{34}\}$, $\{c_{41}, c_{42}, c_{43}, o_{41}, o_{42}\}$
V_4	$\{c_{12}, c_{13}, c_{14}, o_{12}, o_{13}\}$, $\{c_{21}, c_{23}, c_{24}, o_{21}, o_{23}\}$ $\{c_{31}, c_{32}, c_{34}, o_{31}, o_{32}\}$, $\{c_{41}, c_{42}, c_{43}, o_{41}, o_{42}, o_{43}\}$

The circuit computes *majority* commitment for each RSS share using the majority of values input by garblers and checks if there exists a valid opening for each *majority* commitment to retrieve the corresponding input share. If so, the circuit reconstructs the inputs of all parties and then computes the function f which is output to P_4. It is easy to check that the input reconstructed for each honest P_j would indeed correspond to the input shares committed and output by $\mathsf{InputCommit}_j$. This holds since the views of two honest garblers would dictate the *majority* and suffice to reconstruct the committed input of honest P_j. We now consider the cases w.r.t. the corrupt dealer P_k. Suppose $P_k = P_4$ (evaluator) and there is a problem in reconstruction of x_4 (either no majority or appropriate opening). In such a case, the circuit substitutes x_4 with default and outputs f accordingly. Next, suppose corrupt $P_k = P_1$ (garbler). In this case, the circuit may be unable to reconstruct the committed input consistent with the output of $\mathsf{InputCommit}_1()$. For instance, consider corrupt garbler P_1 who distributes the commitment c_{12} to P_3, P_4 and $c'_{12} \neq c_{12}$ to P_2 (c_{12} would be the majority commitment as per $\mathsf{InputCommit}_1$). Now, if the view input by P_1 in $\mathsf{sGod}_4()$ consists of c'_{12}, then majority established by the circuit would be c'_{12} (as the circuit computes majority among versions input by $\{P_1, P_2, P_3\}$ and does not account for the version received by P_4). To handle this and ensure that P_4 gets the same output that the corrupt P_1 would obtain during $\mathsf{sGod}_1()$ (where c_{12} constitutes the majority as per the versions received by $\{P_2, P_3, P_4\}$), the circuit in $\mathsf{sGod}_4()$ additionally does as follows : If a conflict exists amongst the garblers for any value (commitment/opening)

wrt garblers' input, then the circuit outputs the views input by garblers i.e $\{\mathsf{V}_1, \mathsf{V}_2, \mathsf{V}_3\}$. This is also done if the views input by the garblers do not suffice to reconstruct some garbler's input. Both these scenarios occur only when a garbler is corrupt. Privacy against the adversary is thus preserved as the input views are revealed only to the honest evaluator. This trick of making the circuit output views when it is established that one of the garblers is corrupt, enables us to ensure that all parties obtain the output wrt *committed* inputs, as per $\mathsf{InputCommit}_i$ ($i \in [4]$). The formal circuit description appears in Fig 6.1. The idea of augmenting the circuit logic as above is inspired by the 4PC protocol of [IKKP15].

Input and Output: P_i inputs V_i, $i \in [3]$. Circuit outputs $y/\{\mathsf{V}_1, \mathsf{V}_2, \mathsf{V}_3\}$.

Notation: Let $c_{ij}^{(k)}, o_{ij}^{(k)}$ denote $c_{ij}, o_{ij} \in \mathsf{V}_k$ for $i \in [4], j \in \mathsf{ind}(\mathbf{P}_i), k \in [3]$.

Computation: The circuit computes output as follows:

- Initialize $\mathsf{flag}_g = \mathsf{flag}_e = 0$.
- For each $i \in [4], j \in \mathsf{ind}(\mathbf{P}_i), k \in [3]$: Compute c_{ij} as the majority among $\{c_{ij}^{(k)}\}_{k\in[3]}$. If there is no majority, set $\mathsf{flag}_g = 1$ if $i \in [3]$ or $\mathsf{flag}_e = 1$ if $i = 4$. Else, check if there exists an $o_{ij}^{(k)}$ s.t $x_{ij}^{(k)} = \mathsf{Open}(\mathsf{pp}, c_{ij}, o_{ij}^{(k)}) \neq \perp$. If so, set $x_{ij} = x_{ij}^{(k)}$, else set $x_{ij} = \perp$.
- Set $\mathsf{flag}_g = 1$ if there exist $(\mathsf{V}_i, \mathsf{V}_k)$ $i, k \in [3], k \neq i$ s.t V_i and V_k are inconsistent wrt any c_{ij} or o_{ij} where $i \in [3], j \in \mathsf{ind}(\mathbf{P}_i)$.
- Set $\mathsf{flag}_g = 1$ if $x_{ij} = \perp$ for any $i \in [3]$, $j \in \mathsf{ind}(\mathbf{P}_i)$. Else set $x_i = \oplus_j x_{ij}$.
- Set $\mathsf{flag}_e = 1$ and x_4 to default value if $x_{4j} = \perp$ for any $j \in \mathsf{ind}(\mathbf{P}_4)$. Else set $x_4 = \oplus_j x_{4j}$.
- If $\mathsf{flag}_g = 1$, output $\{\mathsf{V}_1, \mathsf{V}_2, \mathsf{V}_3\}$. Else, output $y = f(x_1, x_2, x_3, x_4)$.

Figure 6.1: Circuit Description in sGod_4

The main challenge resulting from the above approach is that each execution sGod_i ($i \in [4]$) needs to be made robust, since each P_i obtains output only during sGod_i (as an evaluator). This is unlike the protocol of [IKKP15] that involves multiple executions of 2-party private-simultaneous messages (PSMs) and achieves robustness based on the fact that a party acts as evaluator in multiple instances, including the one where only honest parties are involved. We choose a variant where all parties are involved in each execution, as it enables us to reduce the number of executions and improve the communication and computational efficiency. Specifically, in terms of the number of GCs that constitute the primary efficiency bottleneck, our (non-optimized) protocol involves 8 GCs as opposed to 12 GCs of [IKKP15]. We now give an overview of a robust single execution $\mathsf{sGod}_4()$: The 2 round subprotocols $\mathsf{InputCommit}_i$ ($i \in [4]$) (Fig 3.1) are run. Next, each garbler computes the GC (for the circuit described in Fig 6.1) and set of commitments on the encoding information (in permuted order) using the shared

randomness and sends this information to P_4 in Round 2. The presence of at least two honest garblers allows P_4 to resolve any ambiguity in this common message of the garblers. Each garbler P_g also transfers the (opening of) encoding information corresponding to its input to the GC i.e V_g. To account for a corrupt garbler, say P_1 who does not send the encoding for V_1 as per the protocol, the garblers additionally do the following: P_2 and P_3 aid in this transfer by sending the encoding corresponding to P_1's wires, for the values in V_1 known to them. Here, we exploit the fact that input variables involved in V_1 are known to either P_2 or P_3 and thereby the transfer of labels corresponding to V_1 is made resilient to P_1's misbehavior. We extend the above trick in a natural way to account even for P_2 and P_3's potential misbehavior.

While the above technique of making a garbler send labels even for its co-garblers' input is effective in achieving robustness against corrupt garbler, the above solution gives rise to the following attack by corrupt P_4: Consider the labels that P_4 gets corresponding to V_1. Wrt the wires corresponding to V_1, P_4 now gets labels not only from P_1 but also from P_2 and P_3 for the values they have in common with V_1. P_4 can exploit this to obtain multiple evaluations of the GC which breaches privacy as demonstrated by the following subtle attack: P_4 distributes c_{41} to P_1, P_2 and c'_{41} to P_3 in Round 1. Upon evaluation of GC using the labels sent by garblers, P_4 would obtain output wrt share x_{41} committed via c_{41} (established as majority commitment by the circuit). However, the additional labels that P_3 sends corresponding to V_1 would be wrt c'_{41}. To obtain another evaluation of f wrt share committed via c'_{41}, P_4 does the following: For input wires involving c_{41} in V_1, use the labels sent by P_3 for V_1. In this evaluation, the input to the circuit would involve c'_{41} in both V_1 and V_3, establish c'_{41} as majority and output accordingly. Note that such attacks can be carried out by corrupt P_4, only if P_4 distributed inconsistent commitments related to her input shares during $\mathsf{InputCommit}_4$. We tackle this security breach as: Corresponding to wires involving P_4's committed shares i.e $\{c_{41}, c_{42}, c_{43}\}$, the (opening of) encoding information is split and garblers send a share of the (opening of) encoding information rather than the entire (opening of) encoding information. We explain how this resolves the problem in context of the example described above – the garblers send the following wrt wires involving c_{41} in V_1: While P_1 sends the encoding of the version of c_{41} received by her, P_2 and P_3 send only a share each of the encoding of the version of c_{41} they possess. In case corrupt P_4 distributed different versions to P_2 and P_3, the shares sent by them would not be consistent and therefore render meaningless to P_4; preventing her from obtaining multiple evaluations. This completes the protocol overview. The formal description of sGod_4 appears in Fig 6.2 and the robust 2-round $\mathsf{2RGodSetup}()$ assuming setup that combines all executions of sGod_i ($i \in [4]$) appears in Fig 6.3.

Inputs and Output: Party P_i has input $x_i, i \in [4]$. P_4 outputs $y = f(x_1, x_2, x_3, x_4)$.

Common Inputs: The circuit Ckt described in Fig 5.1 that takes as input views $\mathsf{V}_1, \mathsf{V}_2, \mathsf{V}_3$ of garblers P_1, P_2, P_3 respectively and computes as output either $y = f(x_1, x_2, x_3, x_4)$ or $\{\mathsf{V}_1, \mathsf{V}_2, \mathsf{V}_3\}$. Let $|\mathsf{V}_i| = \ell$ denote the number of bits in the view V_i $(i \in [3])$

Primitives: A non-interactive commitment scheme (NICOM) $(\mathsf{Com}, \mathsf{Open})$.

Setup: PRF seed r shared amongst garblers $\{P_1, P_2, P_3\}$

Round 1: $P_i, i \in [4]$ runs Round 1 of $\mathsf{InputCommit}_i()$ (Fig 3.1).

Round 2: $P_i, i \in [4]$ runs Round 2 of $\mathsf{InputCommit}_i()$ (Fig 3.1). Besides, the following is done:

- The garblers P_1, P_2, P_3 do the following obtaining all randomness from the common PRF seed r (that was shared during the one-time setup phase):
  - Garble the circuit Ckt as $(\mathsf{GC}, e, d) \leftarrow \mathsf{Gb}(1^\kappa, \mathsf{Ckt})$.
  - Let $p_w \in_R \{0,1\}$ be the permutation bit corresponding to wire index w, $w \in [3\ell]$. Assume $\{e_w^0, e_w^1\}$ be the encoding information for the wire w. Generate commitments to them as: for $b \in \{0,1\}$, compute $(\mathsf{C}_w^b, \mathsf{O}_w^b) \leftarrow \mathsf{Com}(\mathsf{pp}, e_w^{p_w \oplus b})$.
  - P_1,P_2 send $\mathbf{B} = \left\{\mathsf{GC}, \{\mathsf{C}_w^b\}_{w \in [3\ell], b \in \{0,1\}}, d, \{p_w\}_{w \in S}\right\}$ to P_4, where S denotes the set of wire indices involving values in V_4. P_3 sends $\mathsf{H}(\mathbf{B})$ to P_4.
- Let v_g^α $(\alpha \in [\ell])$ denote the value at index α in V_g $(g \in [3])$. Corresponding to each input wire α of V_1 where $\alpha \in [\ell]$, the garblers do the following:
  - P_1 sends $(\mathsf{O}_\alpha^{m_\alpha}, m_\alpha)$ to P_4, where $m_\alpha = p_\alpha \oplus v_1^\alpha$.
  - $P_k, k \in \{2,3\}$ does as follows: If α involves an element in $\mathsf{V}_1 \cap \mathsf{V}_k$, P_k computes $m_\alpha^{(k)} = p_\alpha \oplus v^{(k)}$, where $v^{(k)}$ denotes corresponding value in V_k.
  - For $b \in \{0,1\}$, P_2 and P_3 use common randomness to compute shares $[\mathsf{O}_\alpha^b]_0$ and $[\mathsf{O}_\alpha^b]_1$ of O_α^b such that $\mathsf{O}_\alpha^b = [\mathsf{O}_\alpha^b]_0 \oplus [\mathsf{O}_\alpha^b]_1$.
  - Let I denote the set of common elements $\{c_{41}, c_{42}, c_{43}\}$. If α involves elements in I, P_2 sends $[\mathsf{O}_\alpha^{m_\alpha^{(2)}}]_0$ and P_3 sends $[\mathsf{O}_\alpha^{m_\alpha^{(3)}}]_1$.
  - If α involves elements in $(\mathsf{V}_1 \cap \mathsf{V}_2) \setminus I$, P_2 sends $(m_\alpha^{(2)}, \mathsf{O}_\alpha^{m_\alpha^{(2)}})$. Similarly, if α involves elements in $(\mathsf{V}_1 \cap \mathsf{V}_3) \setminus I$, P_3 sends $(m_\alpha^{(3)}, \mathsf{O}_\alpha^{m_\alpha^{(3)}})$.
- The garblers execute steps analogous to the above for input wires corresponding to V_2 and V_3 as well.

Output Computation by P_4:

- Let $\mathbf{B}_i$ be the version of $\mathbf{B}$ sent by P_i $(i \in [2])$. If $\mathbf{B}_1 = \mathbf{B}_2$, set $\mathbf{B} = \mathbf{B}_1$. Else set $\mathbf{B} = \mathbf{B}_i$ where $\mathsf{H}(\mathbf{B}_i)$ matches the value sent by P_3.
- To retrieve labels corresponding to input wire α of V_1 i.e e_α for $\alpha \in [\ell]$, P_4 does the following: Use $(\mathsf{O}_\alpha^{m_\alpha}, m_\alpha)$ sent by P_1 to obtain $e_\alpha = \mathsf{Open}(\mathsf{pp}, \mathsf{C}_\alpha^{m_\alpha}, \mathsf{O}_\alpha^{m_\alpha})$, where $\mathsf{C}_\alpha^{m_\alpha} \in \mathbf{B}$. If the opening is

invalid, set $\mathsf{Corr}_4 = \{P_1\}$ and do the following.

- If α involves elements in I i.e $\{c_{41}, c_{42}, c_{43}\}$, compute $m_\alpha = p_\alpha \oplus v^{(4)}$, where $v^{(4)}$ denotes the corresponding value in V_4. Compute $\mathsf{O}_\alpha^{m_\alpha}$ as $[\mathsf{O}_\alpha^{m_\alpha^{(2)}}]_0 \oplus [\mathsf{O}_\alpha^{m_\alpha^{(3)}}]_1$ using the shares sent by P_2, P_3 and set $e_\alpha = \mathsf{Open}(\mathsf{pp}, \mathsf{C}_\alpha^{m_\alpha}, \mathsf{O}_\alpha^{m_\alpha})$.
- Else, use $(m_\alpha^{(2)}, \mathsf{O}_\alpha^{m_\alpha^{(2)}})$ if sent by P_2 to retrieve $e_\alpha = \mathsf{Open}(\mathsf{pp}, \mathsf{C}_\alpha^{m_\alpha^{(2)}}, \mathsf{O}_\alpha^{m_\alpha^{(2)}})$. If not obtained from P_2, use $\mathsf{O}_\alpha^{m_\alpha^{(3)}}$ sent by P_3 to retrieve e_α.

- P_4 retrieves other labels corresponding to input wires of $\mathsf{V}_2, \mathsf{V}_3$ similar to the above step. Let $\mathbf{X} = \{e_w\}_{w \in [3\ell]}$, compute $\mathbf{Y} = \mathsf{Ev}(\mathsf{GC}, \mathbf{X})$ where $\mathsf{GC} \in \mathbf{B}$.
- Use $d \in \mathbf{B}$ to decode $\mathbf{Y}$. If $\mathbf{Y}$ decodes to y', output $y = y'$. Else if $\mathbf{Y}$ decodes to $\{\mathsf{V}_1, \mathsf{V}_2, \mathsf{V}_3\}$, compute output y as follows:
  - For $i \in [3]$, check if $\{\mathsf{V}_1, \mathsf{V}_2, \mathsf{V}_3, \mathsf{V}_4\} \setminus \{\mathsf{V}_i\}$ suffices to obtain $\{o_{ij}\}_{j \in \mathsf{ind}(\mathbf{P}_i)}$ such that o_{ij} is a valid opening to c_{ij} output by $\mathsf{InputCommit}_i()$. If so, use o_{ij} to compute x_{ij} and subsequently $x_i = \oplus_j x_{ij}$. Else set x_i to default value.
  - Output $y = f(x_1, x_2, x_3, x_4)$.

Figure 6.2: Single instance with P_4 as evaluator (with setup) sGod_4

Input and Output: P_i inputs x_i and outputs $y = f(x_1, x_2, x_3, x_4)$.

Round 1: Each party P_i executes Round 1 of sGod_i as evaluator and sGod_j ($j \in \mathsf{ind}(\mathbf{P}_i)$) as garbler.

Round 2: Each P_i executes Round 2 of sGod_i as evaluator and sGod_j ($j \in \mathsf{ind}(\mathbf{P}_i)$) as garbler. Output y as the outcome of sGod_i.

Figure 6.3: 2-round GOD (with setup) $\mathsf{2RGodSetup}()$

6.1.1 Optimization

We further provide an optimization technique that reduces the number of GCs involved in $\mathsf{2RGodSetup}()$ to 6 GCs. Each execution sGod_i (Fig 6.2) for $i \in [4]$ requires two garblers to send the GC (specifically $\mathbf{B}$, the set of common information among the 3 garblers) and the third garbler to send $\mathsf{H}(\mathbf{B})$. By considering a field $\mathbb{F}$ with each element of size $\mathbf{B}/2$, we reduce the communication by 25% per execution sGod_i i.e. each sGod_i now requires 1.5 GCs to be communicated, summing up to 6 GCs across the four executions. The optimization trick is as follows: Each garbler interprets the $\mathbf{B}$ computed as concatenation of two elements of $\mathbb{F}$ and $f(x)$ as the linear polynomial with the two elements as coefficients. Each garbler P_g sends $f(x)$ evaluated at a pre-defined point α_g in Round 2. The idea is to allow the evaluator to obtain $\mathbf{B}$ by computing the polynomial $f(x)$ using two honest evaluation points. In order to enable the evaluator to identify these honest evaluation points, we ask each garbler to additionally send a

triple comprising of 3 hash values, namely the hash computed on $f(\alpha_g)$ wrt each of the garblers. The two garblers who sent matching hash triples and correct hash wrt their own evaluation point are identified as honest and are used to interpolate $f(x)$ and subsequently, the set $\mathbf{B}$. The formal description of the optimization appears in Fig 6.4, describing the modifications over the single execution sGod_4 (Fig 6.2).

Notation: Let $|\mathbf{B}| = \ell$. Let $\mathbb{F} = GF(2^{\ell/2})$. Let $\alpha_g, g \in [3]$ is the public point corresponding to garbler P_g.

Primitives: A collision-resistant hash H.

$P_g, g \in [3]$ does the following:

- Run Round 2 of sGod_4 with the following change in transfer of $\mathbf{B}$: Split $\mathbf{B}$ as $\mathbf{B} = a||b$ with $a, b \in \mathbb{F}$. Consider the polynomial $f(x) = ax + b$ of degree 1. Let $\mathcal{H}_1^g = \mathsf{H}(f(\alpha_1)), \mathcal{H}_2^g = \mathsf{H}(f(\alpha_2))$, $\mathcal{H}_3^g = \mathsf{H}(f(\alpha_3))$. Send $\beta_g = f(\alpha_g)$ and $\mathcal{H}_g$ to P_4, where $\mathcal{H}_g = \{\mathcal{H}_1^g, \mathcal{H}_2^g, \mathcal{H}_3^g\}$.

P_4 locally computes $\mathbf{B}$ after Round 2 as follows:

- Identify two garblers P_i and P_j s.t. i) $\mathcal{H}_i = \mathcal{H}_j$, ii) $\mathsf{H}(\beta_i) = \mathcal{H}_i^i$ and iii) $\mathsf{H}(\beta_j) = \mathcal{H}_j^j$. Use β_i, β_j to interpolate $f(x) = ax + b$. Compute $\mathbf{B} = a||b$ to be used for output computation.

Figure 6.4: Optimization of sGod_4

6.2 Correctness and Security of 2RGodSetup

Theorem 3. *Assuming one-way permutations and one-time setup of common randomness pre-shared between every set of 3 among 4 parties, the protocol* 2RGodSetup *(Fig 6.3) securely realizes functionality* $\mathcal{F}_{\mathsf{god}}$ *(Fig 2.1) tolerating one malicious corruption.*

6.2.1 Correctness

Lemma 12. *Protocol* 2RGodSetup() *is correct.*

Proof. We argue that the output y computed by each honest party is based on the unique inputs committed by P_i ($i \in [4]$) in $\mathsf{InputCommit}_i$ (Lemma 1). Consider the output computed by an honest P_4 in sGod_4. As per the protocol, the GC evaluated by P_4 in sGod_4 either leads to output y' or views of the garblers i.e $\{\mathsf{V}_1, \mathsf{V}_2, \mathsf{V}_3\}$. We first consider the case of output y'. It is easy to check that y' must be based on P_i's committed input for honest P_i. This holds since the majority commitments established by the circuit related to x_i are dictated by the views of two honest garblers which must correspond to the committed x_i. Next, we claim that y' is based on the committed input of the corrupt garbler as well, say P_1. To argue this, we

observe that P_1 must have sent consistent commitments to honest garblers $\{P_2, P_3\}$ in Round 1 of $\mathsf{InputCommit}_1$. If not, the GC correctness ensures that the GC outputs $\{\mathsf{V}_1, \mathsf{V}_2, \mathsf{V}_3\}$ (since inputs $\mathsf{V}_2, \mathsf{V}_3$ would be inconsistent wrt P_1's input) which contradicts our assumption that the GC output is y'. It thus holds that the majority commitment established by the circuit wrt P_1's input must be the version held by both P_2, P_3 at the end of Round 1 of $\mathsf{InputCommit}_1$; which is consistent with the output of $\mathsf{InputCommit}_1$. The strong binding property of NICOM ensures the uniqueness of x_1 reconstructed by the circuit. Thus, we conclude that y' is based on the unique inputs committed by P_i ($i \in [4]$) in $\mathsf{InputCommit}_i$. Next, we consider the case when honest P_4 obtains $\{\mathsf{V}_1, \mathsf{V}_2, \mathsf{V}_3\}$ in sGod_4. It is now easy to check that the unique input x_i committed in $\mathsf{InputCommit}_i$ ($i \in [3]$) can be recovered by P_4 by using $\{\mathsf{V}_1, \mathsf{V}_2, \mathsf{V}_3, \mathsf{V}_4\} \setminus \mathsf{V}_i$. This follows directly from the properties of $\mathsf{InputCommit}_i$. Thus, P_4 proceeds to compute the function directly on the committed inputs. □

6.2.2 Security

Before presenting the formal security proof, we begin with a brief intuition of how $\mathsf{2RGodSetup}()$ achieves guaranteed output delivery. Firstly, we argue that an honest evaluator, say P_4 will always be able to evaluate the GC during $\mathsf{sGod}_4()$ robustly which will lead him to obtain the correct output (Lemma 12). Wrt to each honest garbler P_i, P_4 obtains labels corresponding to V_i directly. Now suppose a corrupt garbler, say P_1 does not send labels for V_1. P_4 obtains labels for wires involving $\{c_{41}, c_{42}, c_{43}\}$ in V_1 by using shares sent by P_2 and P_3; both of whom must have obtained the appropriate consistent $\{c_{41}, c_{42}, c_{43}\}$ as distributed by honest P_4 in Round 1. For other wires in V_1, P_4 can simply accept labels sent by P_2 if available, else P_3. We claim that this works even if say P_2 sends labels that do not correspond to P_1's committed input. In such cases, there must be an inconsistency between V_2 and V_3 wrt P_1's input and subsequently the circuit outputs $\{\mathsf{V}_1, \mathsf{V}_2, \mathsf{V}_3\}$; which further enables honest P_4 to obtain the correct output (Lemma 12). We thus conclude that GC evaluation is robust and all honest parties are guaranteed to obtain the correct output.

Finally, we argue that the output obtained by the corrupt party, say P_1 is also based on the unique inputs committed by each P_i ($i \in [4]$) during $\mathsf{InputCommit}_i$. The argument is straightforward- Firstly, corrupt P_1 upon GC evaluation during sGod_1 would obtain y'; specifically the GC would never output the view of the garblers $\{\mathsf{V}_2, \mathsf{V}_3, \mathsf{V}_4\}$. This holds since the garblers $\{P_2, P_3, P_4\}$ are honest and hence, would never be in conflict wrt their input shares. It follows from the earlier arguments that y' would be based on committed inputs of honest parties. Lastly, even the corrupt party's input x_1 reconstructed by the circuit would be the unique value committed in $\mathsf{InputCommit}_1$. This holds since the circuit of sGod_1 would establish the

same majority commitments as $\mathsf{InputCommit}_1$ in case P_1 distributed consistent commitments to at least two among $\{P_2, P_3, P_4\}$ in Round 1 of $\mathsf{InputCommit}_1$. Else, the circuit would substitute x_1 with default which is considered as the unique value committed by P_1 in $\mathsf{InputCommit}_1$ in such a case. This completes the intuition.

We now present the security proof of $\mathsf{2RGodSetup}()$ relative to the ideal functionality $\mathcal{F}_{\mathsf{god}}$ (Fig 2.1). Since the protocol is symmetric, we describe the simulator $\mathcal{S}_{\mathsf{2RGodSetup}}$ for a single case, namely when P_4 is corrupt and the simulator acts on behalf of P_1, P_2, P_3. For better presentation, we present $\mathcal{S}_{\mathsf{2RGodSetup}}$ as a combination of simulators $\mathcal{S}_{\mathsf{sGod}_i}$ $(i \in [4])$ where $\mathcal{S}_{\mathsf{sGod}_i}$ describes the simulation during execution sGod_i. First, we describe simulator $\mathcal{S}_{\mathsf{sGod}_4}$ wrt simulation during single execution sGod_4 where corrupt P_4 acts as evaluator. Next, we describe simulator $\mathcal{S}_{\mathsf{sGod}_1}$ wrt simulation during sGod_1 where corrupt P_4 acts as garbler. The latter is symmetric to simulator $\mathcal{S}_{\mathsf{sGod}_2}$ and simulator $\mathcal{S}_{\mathsf{sGod}_3}$ involving simulation steps during sGod_2 and sGod_3 respectively; both of which involve corrupt garbler P_4.

We give an overview of a couple of technicalities in the simulation. First, simulation of $\mathsf{InputCommit}_i$ $(i \in [4])$ run during $\mathsf{2RGodSetup}()$ is executed as described in figures 4.3 and 4.4. We recall that corrupt P_4's input is extracted by the simulator during $\mathsf{InputCommit}_4$ at the end of Round 1 itself; enabling the simulator to obtain output y via $\mathcal{F}_{\mathsf{god}}$ at the end of Round 1 of simulation itself. Second, we point that wrt executions where P_4 acts as garbler, the adversary corrupting P_4 does not receive any messages from the honest parties as per the protocol. Therefore the steps of $\mathcal{S}_{\mathsf{sGod}_1}$ with P_4 as garbler is quite straightforward. The most interesting case of simulation is during sGod_4 when P_4 acts as evaluator. Here, we use a slight variant of the Yao's privacy simulator that takes as argument not only the output and circuit description but also the encoding information of the input wires. This is needed to enable the simulator acting on behalf of garblers to transfer (possibly) different labels for the same wire corresponding to P_4's input. However, the protocol logic ensures that these labels do not enable P_4 to obtain any additional information beyond the function evaluation on his unique committed set of inputs. Privacy follows from the privacy of garbling scheme, relying on the fact that corresponding to gates involving wires related to honest parties' inputs, P_4 always obtains a single label only. More specifically, the only wires corresponding to which P_4 may obtain both labels are corresponding to gates which take as input only those values which are already known to P_4. Therefore, the intermediate outputs of these gates are values that can be computed by corrupt P_4 locally even in the ideal world. The formal description of the simulators $\mathcal{S}_{\mathsf{sGod}_4}$ and $\mathcal{S}_{\mathsf{sGod}_1}$ appear in Fig 6.5 and Fig 6.6 respectively.

We argue that $\text{IDEAL}_{\mathcal{F}_{\mathsf{god}}, \mathcal{S}_{\mathsf{sGod}_4}} \overset{c}{\approx} \text{REAL}_{\mathsf{sGod}_4, \mathcal{A}}$ when $\mathcal{A}$ corrupts P_4 based on the following series of intermediate hybrids.

HYB_0: Same as $\text{REAL}_{\mathsf{sGod}_4,\mathcal{A}}$.

HYB_1: Same as HYB_0 except that $\{c_{ij}\}_{i\in\mathsf{ind}(\mathbf{P}_4),j\notin\mathsf{ind}(\mathbf{P}_{i4})}$ is replaced with a commitment to a dummy value during $\mathsf{InputCommit}_i$ when P_4 doesn't get access to the corresponding opening information.

HYB_2: Same as HYB_1 except that P_4 is added to $\mathsf{Corr}_k, k \in \mathsf{ind}(\mathbf{P}_4)$ if opening forwarded from P_4 to P_k corresponding to P_i's committed share ($i \in \mathsf{ind}(\mathbf{P}_{4k})$) in $\mathsf{InputCommit}_i$ is different from what was originally committed.

HYB_3: Same as HYB_2, except that garblers P_1, P_2, P_3 use uniform randomness instead of pseudo-randomness.

HYB_4: Same as HYB_3, except that some of the commitments of input wire labels sent on behalf of P_1, P_2, P_3 which will not be opened by P_4 are replaced with commitments on dummy values.

HYB_5: Same as HYB_4 except that the GC is created as $(\mathsf{GC}, \mathbf{X}, d) \leftarrow \mathcal{S}_{\mathsf{prv}}(1^\kappa, \mathsf{Ckt}, \{e_w^0, e_w^1\}_{w\in[3\ell]}, y)$.

Since $\text{HYB}_5 := \text{IDEAL}_{\mathcal{F}_{\mathsf{god}},\mathcal{S}_{\mathsf{sGod}_4}}$, to conclude the proof we show that every two consecutive hybrids are indistinguishable.

$\text{HYB}_0 \overset{c}{\approx} \text{HYB}_1$: The difference between the hybrids is that when the execution results in P_4 not getting access to openings of c_{ij} where $(i \in \mathsf{ind}(\mathbf{P}_4), j \notin \mathsf{ind}(\mathbf{P}_{i4}))$, in HYB_0, c_{ij} is a commitment on x_{ij}, while in HYB_1, it is a commitment on a dummy value. Indistinguishability follows from the hiding of the commitment scheme.

$\text{HYB}_1 \overset{c}{\approx} \text{HYB}_2$: The difference between the hybrids is that in HYB_1 P_4 is added to Corr_k if the opening forwarded by P_4 to P_k in $\mathsf{InputCommit}_i$ $i \in \mathsf{ind}(\mathbf{P}_{4k})$ results in $\perp$, whereas in HYB_2, P_4 is added to Corr_k if the opening sent by P_4 is anything other than what was originally committed. Since the probability with which P_4 could successfully decommit to something other than what was originally committed is negligible (due to the binding of the commitment scheme), the hybrids are indistinguishable.

$\text{HYB}_2 \overset{c}{\approx} \text{HYB}_3$: The difference between the hybrids is that P_1, P_2, P_3 use uniform randomness in HYB_3 rather than pseudorandomness as in HYB_2. The indistinguishability follows via reduction to the security of the PRF.

$\text{HYB}_3 \overset{c}{\approx} \text{HYB}_4$: The difference between the hybrids is that some of commitments of the input wire labels in HYB_3 that will not be opened are replaced with commitments of dummy values in HYB_4. The indistinguishability follows via reduction to the hiding property of the commitment scheme Com.

$\text{HYB}_4 \overset{c}{\approx} \text{HYB}_5$: The difference between the hybrids is in the way $(\mathsf{GC}, \mathbf{X}, d)$ is gener-

ated. In HYB_4, $(\mathsf{GC}, e, d) \leftarrow \mathsf{Gb}(1^\kappa, \mathsf{Ckt})$ is run. In HYB_5, it is generated as $(\mathsf{GC}, \mathbf{X}, d) \leftarrow \mathcal{S}_{\mathsf{prv}}(1^\kappa, \mathsf{Ckt}, \{e_w^0, e_w^1\}_{w\in[3\ell]}, y)$ where $\{e_w^0, e_w^1\}_{w\in[3\ell]}$ are the set of labels corresponding to input wires sampled uniformly at random. Since it holds that P_4 obtains only single label corresponding to all gates involving wires related to honest parties' inputs; indistinguishability follows via reduction to the privacy of the garbling scheme.

$\mathcal{S}_{\mathsf{sGod}_4}$ with corrupt $P_4^\star$

Round 1:

- Simulation of Round 1 of $\mathcal{S}_{\mathsf{InputCommit}_\alpha}$ for $\alpha \in [4]$ (refer figure 4.3, 4.4).
- Invoke $\mathcal{F}_{\mathsf{god}}$ with (Input, x_4) on behalf of P_4 to obtain output y where x_4 denotes the input of P_4 extracted after Round 1 of simulation of $\mathsf{InputCommit}_4$.

Round 2:

- Simulation of Round 2 of $\mathcal{S}_{\mathsf{InputCommit}_\alpha}$ for $\alpha \in [4]$.
- Use uniform randomness to sample the encoding information $\{e_w^0, e_w^1\}_{w\in[3\ell]}$ for the garbled circuit.
- Run $(\mathsf{GC}, \mathbf{X}, d) \leftarrow \mathcal{S}_{\mathsf{prv}}(1^\kappa, \mathsf{Ckt}, \{e_w^0, e_w^1\}_{w\in[3\ell]}, y)$. Choose p_w for $w \in [3\ell]$ uniformly at random.
- For wires related to garblers' input: If w is associated with x_{ij} (such as c_{ij}, o_{ij}) for $i \in [3], 4 \in \mathsf{ind}(\mathbf{P}_{ij})$ (i.e known to P_4^*), then set $m_w = p_w \oplus v$ where v denotes the corresponding bit consistent with the values sent to P_4^* during simulation of $\mathsf{InputCommit}_i$; else choose m_w uniformly at random. Let $\mathsf{C}_w^{m_w}$ be commitments on the entries of $\mathbf{X}$ with $\mathsf{O}_w^{m_w}$ as the corresponding openings. Compute the other commitments on dummy values.
- For wires related to P_4's input: Compute $(\mathsf{C}_w^b, \mathsf{O}_w^b)$ on e_w^b for $b \in \{0, 1\}$ as per the protocol. Set $m_w^{(i)} = p_w \oplus v^{(i)}$ where $v^{(i)}$ denotes the value corresponding to w in V_i (i.e was received on behalf of P_i during $\mathsf{InputCommit}_4$).
- Using the values computed as above, send $\mathbf{B}$ on behalf of P_1, P_2 and $\mathsf{H}(\mathbf{B})$ on behalf of P_3 as per the protocol.
- Corresponding to wire w in V_1 related to garblers' inputs: Send $(m_w, \mathsf{O}_w^{m_w})$ on behalf of P_1. Additionally, send$(m_w, \mathsf{O}_w^{m_w})$ on behalf of P_g $(g \in \{2, 3\})$ if w involves values present in V_g.
- Analogous step to above is executed for wire w in $\mathsf{V}_2, \mathsf{V}_3$ related to garblers' inputs.
- Corresponding to wire w in V_1 related to P_4's input:
  - Send $m_w^{(1)}, \mathsf{O}_w^{m_w^{(1)}}$ on behalf of P_1.
  - If w involves $\{c_{41}, c_{42}, c_{43}\}$, send the share $[\mathsf{O}_w^{m_w^{(2)}}]_0$ and $[\mathsf{O}_w^{m_w^{(3)}}]_1$ on behalf of P_2 and P_3 respectively as per the protocol. Else, send $(m_w^{(g)}, \mathsf{O}_w^{m_w^{(g)}})$ on behalf of P_g $(g \in \{2, 3\})$ if w involves value in $\mathsf{V}_1 \cap \mathsf{V}_g$.
- Analogous step as above carried out for wires in V_2 and V_3 related to P_4's input.

Figure 6.5: $\mathcal{S}_{\mathsf{sGod}_4}$: $\mathcal{S}_{\mathsf{2RGodSetup}}$ during sGod_4

We now argue that $\text{IDEAL}_{\mathcal{F}_{\mathsf{god}}, \mathcal{S}_{\mathsf{sGod}_1}} \overset{c}{\approx} \text{REAL}_{\mathsf{sGod}_1, \mathcal{A}}$ when $\mathcal{A}$ corrupts P_4 based on the following

series of intermediate hybrids.

HYB_0: Same as $\text{REAL}_{\mathsf{sGod}_1,\mathcal{A}}$.

HYB_1: Same as HYB_0 except that $\{c_{ij}\}_{i\in\mathsf{ind}(\mathbf{P}_4),j\notin\mathsf{ind}(\mathbf{P}_{i4})}$ is replaced with a commitment to a dummy value during $\mathsf{InputCommit}_i$ when P_4 doesn't get access to the corresponding opening information.

HYB_2: Same as HYB_1 except that P_4 is added to $\mathsf{Corr}_k, k \in \mathsf{ind}(\mathbf{P}_4)$ if opening forwarded from P_4 to P_k corresponding to P_i's committed share ($i \in \mathsf{ind}(\mathbf{P}_{4k})$) in $\mathsf{InputCommit}_i$ is different from what was originally committed.

HYB_3: Same as HYB_2, except that the honest evaluator P_1 outputs $\perp$ if P_4 sends $\widetilde{\mathsf{O}}_\alpha^{m_\alpha}$ such that $\widetilde{\mathsf{O}}_\alpha^{m_\alpha} \neq \mathsf{O}_\alpha^{m_\alpha}$ but $\mathsf{Open}(\mathsf{pp}, \mathsf{C}_\alpha^{m_\alpha}, \widetilde{\mathsf{O}}_\alpha^{m_\alpha}) \neq \perp$.

Since $\text{HYB}_3 := \text{IDEAL}_{\mathcal{F},\mathcal{S}_{\mathsf{sGod}_1}}$ and $\text{HYB}_0 \overset{c}{\approx} \text{HYB}_2$ follows from argument wrt $\mathcal{S}_{\mathsf{sGod}_4}$, to conclude $\text{IDEAL}_{\mathcal{F},\mathcal{S}_{\mathsf{sGod}_1}} \overset{c}{\approx} \text{REAL}_{\mathsf{sGod}_1,\mathcal{A}}$, it suffices to show that $\text{HYB}_2 \overset{c}{\approx} \text{HYB}_3$. This follows directly from the binding property of the commitment scheme Com.

$\mathcal{S}_{\mathsf{sGod}_1}$ with corrupt P_4^*

Round 1:

- Simulation of Round 1 of $\mathcal{S}_{\mathsf{InputCommit}_\alpha}$ for $\alpha \in [4]$ (refer figure 4.3, 4.4).

Round 2:

- Simulation of Round 2 of $\mathcal{S}_{\mathsf{InputCommit}_\alpha}$ for $\alpha \in [4]$.
- Let r denote the common PRF seed shared between P_2, P_3 and P_4. Use r to compute $\{\mathsf{C}_w^b, \mathsf{O}_w^b\}$ for $b \in \{0,1\}$ and $w \in [3\ell]$ and $\mathbf{B}$ as per the protocol.
- For $\alpha \in [\ell]$ (corresponding to V_1), receive $(\widetilde{\mathsf{O}}_\alpha^{m_\alpha}, m_\alpha)$ on behalf of P_1. Output $\perp$ if $\widetilde{\mathsf{O}}_\alpha^{m_\alpha} \neq \mathsf{O}_\alpha^{m_\alpha}$ but $\mathsf{Open}(\mathsf{pp}, \mathsf{C}_\alpha^{m_\alpha}, \widetilde{\mathsf{O}}_\alpha^{m_\alpha}) \neq \perp$ where $\mathsf{C}_\alpha^{m_\alpha} \in \mathbf{B}$. Else output y on behalf of P_1.

Figure 6.6: $\mathcal{S}_{\mathsf{sGod}_1}$: $\mathcal{S}_{\mathsf{2RGodSetup}}$ during sGod_1

The above argument can be extended to argue indistinguishability of the adversary's view as per simulation of $\mathsf{sGod}_2, \mathsf{sGod}_3$ as well (whose steps are identical to the case of sGod_1). This completes the proof of Theorem 3.

6.3 Without Setup

In this section, we show how the 2-round protocol $\mathsf{2RGodSetup}()$ (non-optimized variant) of Section 6.1 can be modified to construct protocol $\mathsf{2RGod}()$ that avoids setup assumption without inflating the rounds. We recall that the (non-optimized) $\mathsf{2RGodSetup}()$ involves 4 executions, each comprising of 2 GCs. Consider execution $\mathsf{sGod}_4()$ that assumes a shared randomness between the garblers P_1, P_2, P_3. This setup can be avoided in the subprotocol $\mathsf{god}_4()$ of $\mathsf{2RGod}()$ as

follows- Two of the 3 garblers, say P_1, P_2 are assigned to distribute fresh independent randomness, say r_a, r_b respectively to the other garblers in Round 1. In Round 2, wrt both r_a, r_b, the garbler who chooses the randomness transfers the common information $\mathbf{B}$ (comprising of GC and commitment on encoding information e) while the remaining garblers send $\mathsf{H}(\mathbf{B})$. Thus, the number of GCs transmitted is 2 per execution. The transfer of e in Round 2 is similar to $\mathsf{sGod}_4()$, except that now it involves labels corresponding to 2 GCs (one computed using r_a, other using r_b). This completes the description of the messages communicated in $\mathsf{god}_4()$.

We give a quick overview of how the computation by evaluator P_4 in $\mathsf{god}_4()$ is modified according to the changes described above. We observe that the GC wrt which both honest garblers obtain the same randomness in Round 1 can be robustly evaluated. This follows directly from robustness of $\mathsf{sGod}_4()$ and is guaranteed to occur when the garbler acting as randomness distributor is honest. We point that assigning two different garblers to distribute randomness for the two GCs ensures the presence of at least one such GC, whose evaluation and output computation follows similar to $\mathsf{sGod}_4()$. If a corrupt garbler, say P_1 distributes different randomness, say r_a^2 to P_2 and r_a^3 to P_3 in Round 1 and P_1 sends common information $\mathbf{B}_a$ inconsistent with both r_a^2 and r_a^3, she is identified to be corrupt by P_4 and this GC is discarded (only the GC based on r_b distributed by honest garbler is evaluated). However, a major challenge occurs if P_1 sends $\mathbf{B}_a$ consistent with the message of exactly one honest garbler, say P_2 (using r_a^2). Now, P_4 attempts to evaluate the GC based on $\mathbf{B}_a$, wrt which the labels sent by honest P_3 are futile (as they are based on r_a^3). We emphasize that this is a substantial complication over $\mathsf{sGod}_4()$ as $\mathsf{sGod}_4()$ is not equipped to handle cases where communication from honest garbler in Round 2 is meaningless. We handle this as: Since P_1 (randomness distributer) and P_3 are not on the same page wrt $\mathbf{B}_a$, P_4 adds them to her conflict set and identifies P_2 to be honest. P_4 can thereby simply accept the labels sent by P_2 wrt $\mathbf{B}_a$ for all input wires involving values known to P_2. This step ensures that the majority commitments established by the circuit corresponding to honest parties' inputs indeed corresponds to their committed inputs (for instance, in the above context, the version distributed to honest P_2 in Round 1 would be established as majority; since labels given by P_2 would be used wrt both V_2 and V_3). Next, since labels obtained exclusively from P_1 (not from P_2) are related to values known to P_4, P_4 can verify whether they are indeed correct. In case of any problem, P_4 can simply discard this GC and evaluate the GC based on r_b dealt by honest party.

While the above works for labels of honest parties' inputs, a subtle issue emerges when P_2 does not receive input shares consistent with corrupt P_1's *committed* input in Round 1. This is tackled as: Recall that as per $\mathsf{InputCommit}_1$, in the above case, P_4 populates Con_4 with (P_1, P_2). Now, since Con_4 also contains (P_1, P_3), P_4 points P_1 to be corrupt and discards the GC based

on r_a and evaluates only the GC based on r_b distributed by honest P_2. Thus, $\mathsf{god}_4()$ maintains the invariant that wrt the GC whose randomness is dealt by corrupt garbler, either honest P_4 is able to successfully evaluate on committed inputs or the corrupt garbler is exposed leading to P_4 discarding the GC and obtaining output by robust evaluation of other GC (with honest randomness distributor). Further, round 2 of $\mathsf{InputCommit}$ is run (outside of circuit) to enable honest P_4 to compute the output using only one honest garbler's view, when the circuit outputs views. This completes the high-level intuition of robustness of $\mathsf{god}_4()$. The formal description of single execution $\mathsf{god}_4()$ and the 2-round robust $\mathsf{2RGod}()$ that combines all executions appear in Fig 6.7 and Fig 6.8 respectively.

Inputs and Output: Party P_i has input $x_i, i \in [4]$. P_4 outputs $y = f(x_1, x_2, x_3, x_4)$.

Common Inputs: The circuit Ckt described in Fig 5.1 that takes as input views $\mathsf{V}_1, \mathsf{V}_2, \mathsf{V}_3$ of garblers P_1, P_2, P_3 respectively and computes as output either $y = f(x_1, x_2, x_3, x_4)$ or $\{\mathsf{V}_1, \mathsf{V}_2, \mathsf{V}_3\}$. Let $|\mathsf{V}_i| = \ell$ denote the number of bits in the view of each $i \in [3]$

Primitives: A NICOM $(\mathsf{Com}, \mathsf{Open})$.

Round 1: $P_i, i \in [4]$ runs Round 1 of $\mathsf{InputCommit}_i()$ (Fig 3.1). Besides,
- P_1 sends PRF seed r_a to P_2 and P_3. P_2 sends PRF seed r_b to P_1, P_3.

Round 2: $P_i, i \in [4]$ runs Round 2 of $\mathsf{InputCommit}_i()$ (Fig 3.1). Besides,
- The garblers P_1, P_2, P_3 compute $\mathbf{B}_a$, $\mathbf{B}_b$ using PRF seeds r_a, r_b respectively similar to sGod_4.
- P_1 sends $\mathbf{B}_a$. P_2, P_3 send $\mathsf{H}(\mathbf{B}_a)$. P_2 sends $\mathbf{B}_b$. P_1, P_3 send $\mathsf{H}(\mathbf{B}_b)$.
- The garblers execute steps similar to Round 2 of sGod_4 to transfer encoding information wrt both $\mathsf{GC}_a \in \mathbf{B}_a$ and $\mathsf{GC}_b \in \mathbf{B}_b$.

Output Computation by P_4:
- There are 3 cases that occur wrt $\mathbf{B}_a$ sent by P_1:
  (a) Messages of both P_2, P_3 are consistent with $\mathbf{B}_a$: Compute output using $\mathbf{B}_a$ and the corresponding labels sent by the garblers as per output computation in $\mathsf{sGod}_4()$ and terminate.
  (b) Messages of exactly one among P_2, P_3 is consistent with $\mathbf{B}_a$: Populate $\mathsf{Con}_4 = \{P_1, P_j\}$ where P_j $(j \in \{2, 3\})$ denotes the garbler in conflict with P_1 wrt $\mathbf{B}_a$.
  (c) Messages of both P_2, P_3 inconsistent wrt $\mathbf{B}_a$: Set $\mathsf{Corr}_4 = \{P_1\}$.
- Execute steps analogous to above wrt $\mathbf{B}_b$.
- Set $\mathsf{Corr}_4 = P_j$ $(j \in [3])$ if two distinct pairs in Con_4 involve P_j.
- If $\mathsf{Corr}_4 = P_1$ (correspondingly P_2), compute output using $\mathbf{B}_b$ (correspondingly $\mathbf{B}_a$) and the associated labels sent by the garblers as per output computation in $\mathsf{sGod}_4()$ and terminate.
- Suppose Con_4 contains $\{P_1, P_j\}$, garbler $P_k = \{P_1, P_2, P_3\} \setminus \{P_1, P_j\}$ is identified to be honest. P_4 evaluates $\mathsf{GC}_a \in \mathbf{B}_a$ as:

- For each α (involving values known to P_k) where $(\mathsf{O}_\alpha^{m_\alpha}, m_\alpha)$ is obtained from P_k, compute $e_\alpha = \mathsf{Open}(\mathsf{pp}, \mathsf{C}_\alpha^{m_\alpha}, \mathsf{O}_\alpha^{m_\alpha})$, $\mathsf{C}_\alpha^{m_\alpha} \in \mathbf{B}_a$.
- For α involving elements in V_4: Compute $m_\alpha = p_\alpha \oplus v^{(4)}$, where $v^{(4)}$ denotes the corresponding value in V_4.
- For α corresponding to input V_j of the GC, do the following if α involves $\{c_{41}, c_{42}, c_{43}\}$: Obtain a share each of $\mathsf{O}_\alpha^{m_\alpha}$ from P_1 and P_k; combine the shares to compute $\mathsf{O}_\alpha^{m_\alpha}$, and subsequently e_α using $\mathsf{C}_\alpha^{m_\alpha} \in \mathbf{B}_a$. If the opening is invalid, set $\mathsf{Corr}_4 = \{P_1\}$.
- Using the opening of encoding information $\mathsf{O}_\alpha^{m_\alpha}$ sent by P_1 wrt wires α for which e_α has not been computed yet (values in $\mathsf{V}_4 \setminus \mathsf{V}_k$), compute $e_\alpha = \mathsf{Open}(\mathsf{pp}, \mathsf{C}_\alpha^{m_\alpha}, \mathsf{O}_\alpha^{m_\alpha})$, where $\mathsf{C}_\alpha^{m_\alpha} \in \mathbf{B}_a$. If any opening is invalid, set $\mathsf{Corr}_4 = P_1$ and compute output using $\mathbf{B}_b$ and the associated labels as per output computation in $\mathsf{sGod}_4()$. Else, use $\mathbf{X} = \{e_\alpha\}_{\alpha \in [3\ell]}$ to compute $\mathbf{Y} = \mathsf{Ev}(\mathsf{GC}_a, \mathbf{X})$. Decode $\mathbf{Y}$ to compute output similar to steps of sGod_4 (using information obtained in Round 2 of $\mathsf{InputCommit}_i$ $(i \in [4])$ additionally if GC outputs views).

- If output has not been computed, evaluate $\mathsf{GC}_b \in \mathbf{B}_b$ by following steps analogous to the above in order to obtain output.

Figure 6.7: Single garble instance $\mathsf{god}_4()$

Input and Output: P_i inputs x_i and outputs $y = f(x_1, x_2, x_3, x_4)$.

Round 1: Each party P_i executes Round 1 of $\mathsf{god}_i()$ as evaluator and $\mathsf{god}_j()$ $(j \in \mathsf{ind}(\mathbf{P}_i))$ as garbler.

Round 2: Each P_i executes Round 2 of god_i as evaluator and sGod_j $(j \in \mathsf{ind}(\mathbf{P}_i))$ as garbler. Output y as the outcome of god_i.

Figure 6.8: 2-round GOD 2RGod

6.4 Correctness and Security of 2RGod

Theorem 4. *Assuming one-way permutations exist, the protocol* 2RGod *(Fig 6.8) securely realizes functionality* $\mathcal{F}_{\mathsf{god}}$ *(Fig 2.1) tolerating one malicious corruption.*

6.4.1 Correctness

Lemma 13. *Protocol* $\mathsf{2RGod}()$ *is correct.*

Proof. We argue that the output y computed by each honest party is based on the unique inputs committed by P_i $(i \in [4])$ in $\mathsf{InputCommit}_i$ (Lemma 1). Consider the output computed by an honest P_4 in god_4. Since the proof follows directly from correctness of 2RGodSetup (Lemma 12)

in case P_4 evaluates the GC whose randomness is distributed by honest garbler, we present the argument for the case when P_4 evaluates GC based on r_a, distributed by corrupt garbler, say P_1. As per the protocol, this GC would be evaluated only if P_1 distributed r_a to atleast one among P_2, P_3, say P_2 alone. (If consistent r_a was distributed to both P_2, P_3, correctness follows from Lemma 12). As per the protocol, the GC upon evaluation would either lead to output y' or views of the garblers i.e $\{\mathsf{V}_1, \mathsf{V}_2, \mathsf{V}_3\}$.

We first consider the case of output y'. It is easy to check that honest P_4 would proceed to evaluation of this GC only if he obtained labels corresponding to his committed input. Next, we argue that y' must be based on committed inputs of honest P_2 and P_3 as well - since P_2's view is consistent with the unique value committed at the end of $\mathsf{InputCommit}_2$ and $\mathsf{InputCommit}_3$, any misbehavior by P_1 wrt shares of x_2 or x_3 would lead to the circuit outputting $\{\mathsf{V}_1, \mathsf{V}_2, \mathsf{V}_3\}$ (as there is a conflict wrt garbler's input). This is a contradiction to our assumption that circuit outputs y'. Finally, we analyze the case of corrupt P_1's input. Note that since circuit did not output $\{\mathsf{V}_1, \mathsf{V}_2, \mathsf{V}_3\}$, the x_1 reconstructed by the circuit must be consistent with P_2's view i.e the version of commitments on shares that P_1 gave to P_2 in Round 1 of $\mathsf{InputCommit}_1$. We claim that this must consistent with the unique value committed by P_1 in $\mathsf{InputCommit}_1$ - If not, then the majority commitment related to P_1's input should be present with P_3 and P_4. In such a case, P_4 would have populated Con_4 with $\{P_1, P_2\}$ (as the versions of commitment on P_1's shares obtained from P_1, P_2 would differ). Further, recall that since P_3 did not send GC consistent with r_a, P_4 must have populated $\mathsf{Con}_4 = \{P_1, P_3\}$ as per the protocol. Consequently, P_4 would have concluded that P_1 is corrupt and discarded this GC. We have arrived at a contradiction to our assumption that P_4 evaluates the GC based on r_a. Thus, we conclude that y' is based on the unique inputs committed by P_i ($i \in [4]$) in $\mathsf{InputCommit}_i$.

Next, we consider the case when honest P_4 obtains $\{\mathsf{V}_1, \mathsf{V}_2, \mathsf{V}_3\}$ in god_4. Here, we point that unlike in sGod_4, P_4 is assured to get the view of only one honest garbler, P_2 (in the context of the above example). It is easy to check that P_4 can compute the committed input of the honest parties since the views of two honest parties at the end of Round 1 itself suffices to compute the committed input of all honest parties. Lastly, we analyze how P_4 is able to retrieve P_1's committed input. Note that as per the argument above (about why y' is based on committed x_1), it is evident that P_4 must have evaluated this GC only if the unique input committed by P_1 during $\mathsf{InputCommit}_1$ is available to P_2 at the end of Round 1. Thus, P_4 can compute the committed shares x_{13}, x_{14} using V_2 output by the GC. Wrt share x_{12} not held by P_2 but held by P_3, note that it follows from the description of $\mathsf{InputCommit}_1$ that atleast one among P_3, P_4 must have obtained the opening corresponding to committed x_{12} at the end of Round 1 itself and forwarded it to the other in Round 2. We can thus conclude that P_4 has access to the

committed value of x_{12} at the end of Round 2, thereby enabling him to compute the unique value x_1 committed in $\mathsf{InputCommit}_1$. Thus, P_4 proceeds to compute the function directly on the committed inputs. This completes the proof of correctness. □

6.4.2 Security

In this section, we present a sketch of the simulation proof of 2RGod relative to the ideal functionality $\mathcal{F}_{\mathsf{god}}$ (Fig 2.1). Similar to $\mathcal{S}_{\mathsf{2RGodSetup}}$, we describe the simulator $\mathcal{S}_{\mathsf{2RGod}}$ for a single case, namely when P_4 is corrupt and the simulator acts on behalf of P_1, P_2, P_3 and consider $\mathcal{S}_{\mathsf{2RGod}}$ to be a combination of $\mathcal{S}_{\mathsf{god}_i}$ ($i \in [4]$) where $\mathcal{S}_{\mathsf{god}_i}$ describes the simulation during execution god_i.

We begin with description of simulator $\mathcal{S}_{\mathsf{god}_4}$ wrt simulation during god_4 where corrupt P_4 acts as evaluator. Note that in this case, both randomness distributors are constituted by honest garblers, on behalf of whom the simulator acts. Thereby, the simulation proceeds identical to $\mathcal{S}_{\mathsf{sGod}_4}$ except that in Round 2, there are two copies of GC and encoding information that need to be simulated. We can thus conclude based on the security arguments of $\mathcal{S}_{\mathsf{sGod}_4}$ that the view of $\mathcal{A}$ corrupting P_4 in ideal execution of god_4 is indistinguishable from the real execution of god_4.

Next, we describe the simulator $\mathcal{S}_{\mathsf{god}_1}$ wrt simulation during god_1 where corrupt P_4 acts as garbler. If P_4 does not act as randomness distributor, it is easy to check that the simulation can proceed similar to $\mathcal{S}_{\mathsf{sGod}_1}$ (except involving two instances of GC and encoding information). Now suppose P_4 acts as a designated randomness distributor wrt common information $\mathbf{B}_a$. On behalf of honest P_1, $\mathcal{S}_{\mathsf{god}_1}$ checks if P_4 sent $\mathbf{B}_a$ consistent with randomness received on behalf of atleast one among P_2, P_3. If not, simulator terminates with output y (obtained upon invoking $\mathcal{F}_{\mathsf{god}}$ with x_4 extracted at end of Round 1 of $\mathsf{InputCommit}_4$). Else, $\mathcal{S}_{\mathsf{god}_1}$ populates the corrupt and conflict set on behalf of honest P_1 as per the protocol. If $\mathcal{C}_1 = \varnothing$, $\mathcal{S}_{\mathsf{sGod}_1}$ checks if the (opening of) labels obtained from P_4 on behalf of honest P_1, that are to be used for evaluation as per the protocol god_1, indeed correspond to the same (opening of) labels as derived from the corresponding randomness (known on behalf of atleast one among P_2, P_3 wrt whom $\mathbf{B}_a$ sent by P_4 is consistent). If not, but the opening is valid wrt corresponding commitment, $\mathcal{S}_{\mathsf{god}_1}$ outputs $\perp$ on behalf of honest P_1. As argued in $\mathcal{S}_{\mathsf{sGod}_1}$, this can be reduced to violating the binding property of commitment scheme; which occurs only with negligible probability. We can thus conclude that the view of $\mathcal{A}$ corrupting P_4 in ideal execution of god_1 is indistinguishable from the real execution of god_1. The simulators $\mathcal{S}_{\mathsf{god}_2}$, $\mathcal{S}_{\mathsf{god}_3}$ are analogous to $\mathcal{S}_{\mathsf{god}_1}$ as they all deal with corrupt garbler P_4. This completes the security proof of 2RGod.

Chapter 7

Experimental Results

In this section, we provide elaborate results of our implementation. We use the circuits of AES 128 and SHA 256 as benchmark circits. We begin with the description of the setup in terms of hardware, software, local area network (LAN), wide area network (WAN) and provide a detailed comparison with the relevant state-of-the-art.

Hardware Our experiments are demonstrated both in LAN and WAN setting as they are better suited for high latency networks. For LAN setting, our system specifications include a 32GB RAM, an Intel *i*7 octa-core processor with 3.6GHz processing speed. For WAN setting, we utilize instances of Microsoft Azure D4s_v3 that are located in West US, South India, East Australia and East Japan. As our protocols cater to systems with limited bandwidth support, we limit the bandwidth for the WAN setting to 8Mbps. The slowest link has the round trip time of 0.21 s between West US and South India. Our hardware has support from AES-NI instructions.

Software The operating system used for both LAN and WAN setting is *ubuntu* 16.04LTS (64-bit). Our code is built as per the standards of C++11. We rely on the libgarble library that is built from JustGarble library for the code of garbled circuits. Additionally, we instantiate our garbling scheme with the best optimization of half-gates [ZRE15]. We instantiate our commitment scheme with hash for empirical purposes. A multi-threaded environment is created for improved efficiency. The parties emulate a complete-graph network and sockets are used to communicate the data. All our results depict the average values taken for 20 runs of each experiment.

Comparison We compare our results with the state-of-the-art 3PC and 4PC of [MRZ15, IKKP15, BJPR18]. We have implemented all these protocols in our environment for unified comparison. We do not compare with 5PC of [CGMV17] as it relies on distributed garbling

to tackle 2 corruptions which incurs huge cost even for weaker *abort* security owing to larger circuits. Tables 7.1, 7.2 indicate the performance of our 3RFair and 3RGod when compared to that of [MRZ15, BJPR18]. The tables depict the values separately for the asymmetric roles of garbler and evaluator. However, each party in 2RGod and [IKKP15] emulates symmetric roles acting as garbler and as evaluator for the same number of instances. Also, we need to aptly compare the efficiency of constructions with different number of parties (3PC and 4PC). Hence, Table 7.3 depicts the average performance *per party* for all protocols. In all tables, the bracketed values indicate the worst case run of 3RGod. The tables depict the computation time (**CT**), runtime (CT + network time) for LAN (**LAN**) and runtime in WAN (**WAN**). WAN runtime indicates the influence of round complexity and communication of the protocols, given the proximity of servers.

The performance of our 3-round 3RFair is noteworthy where the overall overhead incurred over the selective abort 3PC of [MRZ15] is a nominal value in the range $1.14 - 1.29$ s (range taken over both circuits) in terms of WAN. Despite using 2 GCs, the protocol 3RFair incurs only an overhead of < 23 ms in LAN (majority due to computation time) over that of 3PC fair [BJPR18], while saving one round of interaction that translates to a gain of at most 0.34 s for garbler and 0.07 s for evaluator role in WAN and improves for larger circuits as exhibited in Table 7.1, 7.2. Moreover, the protocols 3RFair and 3RGod (inclusive of the worst case) improve over the state-of-the-art 5-round 4PC GOD in high-latency network albeit the use of an additional GC. The saving amounts to best case saving of 0.44 s in WAN. The 4PC GOD of [BJPR18] exploits an almost idle party in the system, while our 3RGod protocol involves symmetry with two garblers and two evaluators, hence the resulting efficiency of 3RGod in WAN highlights its practical worth. However, the overhead in computation and communication is due to the use of 2 and 4 GCs in 3RFair and 3RGod respectively compared to the use of a single GC in 4PC GOD of [BJPR18]. Like the authors of [BJPR18], we do not implement the 3PC GOD of [BJPR18] in WAN as it requires a robust channel. Moreover, the per-party gain in WAN of our 3-round protocols over 3PC fair of [BJPR18] (4 rounds) naturally implies gain over 3PC GOD of [BJPR18] (5 rounds with additional broadcast).

In terms of per party comparison, both our 3-round 3RFair and 3RGod gain appx $0.02-0.28$ s over the 4-round 3PC fair of [BJPR18] in terms of WAN runtime, while achieving stronger security notions. This gain comes from the presence of an additional honest party in the computation in 4PC. Additionally, the performance of 3RGod is not far from the selective abort protocol of [MRZ15] and incurs an overhead of $0.14 - 1.79$ ms, $0.05 - 0.22$ s per party in terms of LAN and WAN respectively. The empirical values for the round-optimal GOD protocols of [IKKP15] and ours indicate that the saving of 4 GCs is significant for real-time systems(0.5 s saving in WAN).

Table 7.1: Computation time (**CT**), Runtime for LAN, WAN (bandwidth 8Mbps), Communication (**CM**) of all protocols for **AES** ($g \in [2]$, $e \in \{3,4\}$).

Protocol	**Setting, Security**	**CT(ms)**		**LAN(ms)**		**WAN(s)**		**CM(KB)**	
		P_g	P_e	P_g	P_e	P_g	P_e	P_g	P_e
[MRZ15]	3PC Abort	1.45	0.91	1.54	1.45	0.58	0.56	153.2	2.25
[BJPR18]	3PC fair	1.37	0.92	1.48	1.43	0.85	0.64	161.55	2.27
[BJPR18]	3PC GOD	1.57	1.36	1.76	1.61	–	–	153.39	2.29
[BJPR18]	4PC GOD	1.44	0.87	1.95	1.48	0.84	0.87	163.31	8.1
3RFair	4PC Fair	1.71	0.76	1.97	1.06	0.72	0.78	312.58	10.0
3RGod	4PC GOD	1.69	1.08	1.95	1.34	0.71	0.76	312.58	10.0
		(+0.72)	(+1.05)	(+0.86)	(+1.28)	(+0.06)	(+0.05)	(+2.3)	(+ 156.29)

Table 7.2: Computation time (**CT**), Runtime for LAN, WAN (bandwidth 8Mbps) and Communication (**CM**) of all protocols for **SHA** where $g \in [2]$ and $e \in \{3,4\}$. (The 4th almost idle party in 4PC GOD of [BJPR18] has the following values for AES,SHA in order: CT=0.04 ms,0.09 ms; LAN=0.23 ms,0.6 ms; WAN=0.42 s,0.84 s; Comm=2.1 KB,2.1 KB).

Protocol	**Setting, Security**	**CT(ms)**		**LAN(ms)**		**WAN(s)**		**CM(KB)**	
		P_g	P_e	P_g	P_e	P_g	P_e	P_g	P_e
[MRZ15]	3PC Abort	13.34	10.1	16.72	15.96	1.0	0.96	3073.65	4.51
[BJPR18]	3PC fair	13.92	9.7	16.89	15.85	1.32	1.12	3089.7	4.52
[BJPR18]	3PC GOD	14.86	10.66	17.48	16.52	–	–	3074.19	4.68
[BJPR18]	4PC GOD	13.97	10.81	17.68	16.72	1.23	1.28	3091.9	14.26
3RFair	4PC Fair	19.14	10.16	21.7	14.43	0.98	1.05	6157.5	18.13
3RGod	4PC GOD	19.17	10.18	21.73	14.56	0.97	1.08	6157.5	18.13
		(+11.3)	(+13.24)	(+0.12)	(+0.10)	(+0.11)	(+0.09)	(+9.06)	(+3078.82)

Table 7.3: Average per party values of Computation time (**CT**), Runtime for LAN, WAN (bandwidth 8Mbps) and Communication (**CM**) for all protocols.

Protocol	**Setting, Security**	**CT(ms)**		**LAN(ms)**		**WAN(s)**		**CM(MB)**	
		AES	**SHA**	**AES**	**SHA**	**AES**	**SHA**	**AES**	**SHA**
[MRZ15]	3PC Abort	1.27	12.26	1.51	16.46	0.57	0.98	0.10	2
[BJPR18]	3PC fair	1.22	12.51	1.46	16.54	0.78	1.3	0.10	2.01
[BJPR18]	3PC GOD	1.5	13.46	1.71	17.16	–	–	0.10	2
[BJPR18]	4PC GOD	0.95	9.71	1.41	9.83	0.74	1.14	0.08	1.51
[IKKP15]	4PC GOD	6.49	72.54	7.92	83.68	1.74	2.6	10.98	28.08
3RFair	4PC Fair	1.23	14.65	1.52	18.06	0.75	1.02	0.16	3.02
3RGod	4PC GOD	1.38	14.67	1.65	18.14	0.74	1.03	0.16	3.02
		(+0.88)	(+12.27)	(+1.07)	(+0.11)	(+0.05)	(+0.1)	(+0.07)	(+1.51)
2RGod	4PC GOD	6.28	71.72	7.93	83.243	1.58	2.09	10.68	21.96

Further, for huge circuits, having a one-time setup, while obtaining communication efficient (reduction by half) and round optimal protocol (2RGodSetup) goes a long way in improving the performance. Thus, our two round protocols strike a good balance between round optimality and efficiency.

It is observed that, the performance of our protocols in high latency networks improves with the size of circuits. Also, the difference in the latency and communication between AES and SHA circuits reflects the circuit size. Lastly, as bandwidth increases, the gap in performance of 2-round protocols and their efficient counterparts closes in (although computation is still

higher), owing to the amount of data that the channels can carry at once. In essence, for low-bandwidth systems which is the highlight of the paper, the overheads for our 3 round protocols is owed to the use of more GCs. This overhead, however, is annulled by the gain resulting from fewer rounds of interaction, thus bridging the gap between efficiency and optimal round complexity of 2, which is the foremost need of networks such as the Internet.

Part II

Beyond Honest Majority: 4PC in Best-of-Both-Worlds Setting

Chapter 8

Introduction

MPC protocol comes in distinct flavours with varying degree of robustness– guaranteed output delivery, fairness and unanimous abort. The strongest security, *guaranteed output delivery*, implies that all parties are guaranteed to obtain the output, regardless of the adversarial strategy. In the weaker notion of *fairness*, the corrupted parties receive their output if and only if all honest parties receive their output. In the further weaker guarantee of *unanimous abort*, fairness may be compromised, yet the adversary cannot break unanimity of honest parties. That is, either all or none of the honest parties receive the output. While highly sought-after, the former two properties can only be realised, when majority of the involved population is honest [Cle86]. In the absence of this favorable condition, only unanimous abort can be attained. With these distinct affordable goals, MPC with honest majority [BGW88, CCD88, RB89, BMR90, Bea91, DN07, ACGJ18] and dishonest majority [GMW87, DO10, GGHR14, BHP17, ACJ17, HHPV18, BGJ+18] mark one of the earliest demarcations in the world of MPC.

With complementary challenges and techniques, each setting independently stands tall with spectacular body of work. Yet, the most worrisome shortcoming of these generic protocols is that: a protocol in *one* setting completely breaks down in the *other* setting i.e. the security promises are very rigid and specific to the setting. For example, a protocol for honest majority might no longer even be "private" or "correct" if half (or more) of the parties are corrupted. A protocol that guarantees security with unanimous abort for arbitrary corruptions cannot pull off the stronger security of guaranteed output delivery or fairness even if only a "single" party is corrupt. The quest for attaining the best feasible security guarantee in the respective settings of honest and dishonest majority in a *single* protocol sets the beginning of a brand new class of MPC protocols, termed as *Best of Both Worlds* (BoBW) [IKLP06, Kat07, IKK+11]. In critical applications such as voting [KMO01, NBK15], secure auctions [DGK09], secure aggregation

[BIK+17], federated learning and prediction [MR18, MZ17], financial data analysis [BTW12] and many more, where privacy of the inputs of an honest party needs protection at any cost and yet a robust completion is called for (as much as theoretically feasible), BoBW MPC protocols are arguably the best fit. Having discussed the motivation and the security expectations, we now highlight the known feasibility results and give an outline of our constructions.

On the feasibility of BoBW MPC: Denoting the threshold of corruption in honest and dishonest majority case by t and s respectively, an ideal BoBW MPC should promise the best possible security in each corruption scenario for any population of size n, as long as $t < n/2$ and $s < n$. Specifically, an ideal protocol in BoBW model would simultaneously achieve GOD against $t < n/2$ active corruptions and security with unanimous abort (UA) against $s < n$ active corruptions. Quite contrary to the expectation, [Kat07, IKK+11] show that ideal BoBW is impossible to achieve in expected polynomial time (in the security parameter) when the sum of corruption thresholds in honest and dishonest majority exceeds (or equals) n. Thus, the feasibility reduces to achieving GOD against t active corruptions and UA against s active corruptions under the constraint that $t + s < n$. In the world of 4 parties (4PC), we provide the first efficient construction of a BoBW protocol that promises GOD against $t = 1$ active corruption and UA against $s = 2$ active corruptions, optimally respecting the feasibility.

A number of relaxations were proposed to get around the impossibility result. We consider the most meaningful relaxation provided in the work of [LRM10], where the best possible security of guaranteed output delivery is compromised to the second-best notion of fairness in the honest-majority setting. This notion brings back the true essence of BoBW protocols with no constraint on n, apart from the natural bounds of $t < n/2$ and $s < n$. Furthermore, fairness is almost as good as guaranteed output delivery for many practical applications where the adversary is rational enough and does not wish to fail the honest parties at the expense of losing its own output. In this direction, we provide the first efficient 4PC BoBW promising fairness against $t = 1$ active corruption and UA against $s = 3$ active corruptions. We also provide a simpler extension for this BoBW notion for 3 parties promising fairness against $t = 1$ active corruption and UA against $s = 2$ active corruptions.

We consider 4PC as 4 is the least number of parties which offers meaningful yet challenging security notions for BoBW model, while adhering to the feasibility. To elaborate, for 3-party setting (3PC), the BoBW feasibility $t + s < n$ to attain GOD against t corruptions *and* UA against s corruptions reduces to either simply GOD for honest majority ($t = 1, s = 1$) [BJPR18, PR18] *or* UA for dishonest majority ($t = 0, s = 2$) [CKMZ14], both of which have known practical constructions as cited.

8.1 Related Work

In this section, we talk about the relaxations that were proposed to get around the impossibility result of [IKK+11, Kat07]. As discussed already, in the work of [LRM10], the security expectation for honest-majority setting is compromised to fairness instead of GOD. Allowing the adversary to learn s evaluations of the function in the dishonest-majority case, [IKLP06] shows yet another way to circumvent the impossibility result. While the standard definition of security restricts to a single evaluation of the the ideal functionality, this weakening corresponds to the adversary accessing the functionality s times, each time with distinct inputs *exclusively* corresponding to the corrupt parties. Another circumvention comes at the expense of achieving a weaker notion of $O(1/p)$-security with abort in the dishonest-majority case. Roughly speaking, this notion means that the actions of any polynomial-time adversary in the real world can be simulated by a polynomial-time adversary in the ideal world such that the distributions of the resulting outcomes cannot be distinguished with probability better than $O(1/p)$. [IKLP06] shows yet another circumvention by weakening the adversary in dishonest-majority case from active to passive. Appealing to security against non-rushing adversary (equivalently, assuming simultaneous message transmission) in order to break the polynomial-round barrier, [Kat07] gets partial success and improves the impossibility bound to logarithmic, without any matching upper bound which eludes till date. On the contrary, constructions are known when $t+s<n$ is assumed [IKLP06], tolerating active and rushing corruptions and giving best possible security in both the honest and dishonest majority case.

A more fine-grained graceful degradation of security is dealt with in the works of [LRM10, HLMR11, HLMR12, HLM13] considering a mixed adversary that can simultaneously corrupt in both active and semi-honest style. An orthogonal notion of BoBW security is considered in [Cha89, HMZ08, LRM10] where information-theoretic and respectively computational security is the desired goal in honest and dishonest majority setting. Indeed, information-theoretic security is another note-worthy trade-mark of the honest-majority MPC protocols and in the dishonest-majority regime, computational security is the default choice [FHW04, Cha89]. [LRM10] is by far the only work that considers attaining guaranteed output delivery or fairness with information-theoretic security in the honest-majority world and unanimous abort with computational security otherwise. [HIKR18] puts forward a notion of graceful degradation of security purely in the context of information-theoretic security with the new conceptual contribution of a non-traditional yet meaningful notion of information-theoretic security against dishonest majority. They show their notion to be the best possible information-theoretic guarantee achievable in the honest and dishonest majority simultaneously and further demonstrate

realization of a specific class of MPC functions. [GGR18] studies the communication efficiency in the BoBW setting.

8.2 Our Contribution

In the regime of MPC with small population, we investigate the adversarial model beyond the traditional honest majority and dishonest majority in an efficient manner suitable for practical systems. Specifically, this is the first work to explore the efficiency of constant round protocols in *Best of Both Worlds* [IKLP06] – simultaneously achieving varying security notions based on honest/dishonest majority. Both the protocols are based on distributed garbling (DG) and are realized for $n = 4$ parties. The first construction, $\pi_{\mathsf{bobw.god}}$, simultaneously achieves *guaranteed output delivery* (GOD) for $t = 1$ active corruption and *unanimous abort* (UA) for $s = 2$ active corruptions while respecting the constraint of $t + s < n$ as per [IKLP06]. Further, relaxing the honest-majority security from GOD to fairness to eliminate the above constraint as in [LRM10], the second construction $\pi_{\mathsf{bobw.fair}}$ simultaneously achieves *fairness* for $t = 1$ active corruption and *unanimous abort* for $s = 3$ active corruptions. We also present a simpler 3PC variant of this protocol for $t = 1$ and $s = 2$. Below, the prime technical contributions are summarized.

The heart of $\pi_{\mathsf{bobw.god}}$ protocol lies in putting together the numerous tools optimized specially for small parties in a way that the construction stands secure for both worlds of honest-majority and dishonest-majority. The key ideas of construction come from– (1) Enabling the honest parties to identify the corruption scenario (honest / dishonest majority) in some cases and proceed accordingly, (2) Secret-sharing each input amongst the remaining 3 parties and exploiting the existence of two owners of each input-share to attain robustness when needed. (3) Using the inexpensive tricks of seed-distribution (SD) [CGMV17] and attested OT (AOT) [CGMV17] to handle malicious adversary despite relying only on passively secure DG scheme. (4) Culmination of the tools of semi-honest DG, AOT and SD to tackle one malicious corruption and topping it with the technique of cut-and-choose [Lin16] to undo the possible damage caused by a 2-party coalition. (5) Identification and exclusion of a misbehaving party and proceeding to run an efficient 3PC [MRZ15] (modified to achieve UA) amongst the remaining parties. Note that the promise of UA from 3PC is sufficient since if $t = 1$, then the sole corrupt party has already been eliminated and the remaining 3 honest parties naturally achieve GOD in 3PC. If $s = 2$, then the 3PC is run in the face of *one* corruption promising at most UA security as desired. Lastly, maintaining input privacy and consistency in all corruption scenarios is also crucial and aptly handled by our construction.

For $\pi_{\mathsf{bobw.fair}}$, we rely on a 4-party DG scheme maliciously secure against 3 corruptions in the abort model along with tools such as Tiny OT [NNOB12] for authentication purposes and

replicated secret sharing (RSS) for ensuring BoBW guarantees. We instantiate the DG with the efficient state-of-the-art scheme of [WRK17]. To tackle fairness violation for the case of $t = 1$ when the adversary does not engage in the output construction (but computes the output based on messages from honest parties), we adopt 3-party, 1-private RSS of the output-mask shares of the underlying garbled circuit along with authentication on these re-shares. This step involves non-trivial challenges to ensure valid authentication of the re-shares w.r.t. all parties as the authentication scheme is pairwise and not publicly verifiable. The purpose of 3-party, 1-private RSS as opposed to simpler additive sharing is to ensure unanimity for the case of an adversary corrupting 2 parties i.e. to ensure that both the honest parties are in agreement of the output obtained. Additionally, we scale the construction to 3PC with $t = 1, s = 2$ by providing a simpler variant of $\pi_{\mathsf{bobw.fair}}$. Using the underlying DG scheme as a blackbox, our constructions with stronger BoBW guarantees incur minimal overhead and are highly efficient as exhibited by empirical results.

Empirical Results and Comparison We provide a detailed empirical analysis of our protocols in Chapter 13. The implementation results include realization of AES-128 and SHA-256 circuits in both LAN and WAN setting for $\pi_{\mathsf{bobw.fair}}$. However, to show the practicality of $\pi_{\mathsf{bobw.god}}$ in critical systems, we realize it with the widely-used voting system.

8.3 Outline of Part II

Post the introduction, this part of the thesis starts with the preliminaries required for both BoBW constructions in Chapter 9 followed by the building blocks in Chapter 10. Chapter 11 presents the details of the BoBW construction achieiving GOD in honest majority. Chapter 12 presents the details of the BoBW construction achieving fairness in honest majority. Both protocols are backed with a detailed security proof presented via the existence of a simulator. The final chapter provides elaborate implementation results. The results for the fair protocol show that our protocol incurs a minimal overhead to provide BoBW guarantees over the protocol of [WRK17] on which it is built while the results for GOD protocol are shown via implementation of the application of voting.

Chapter 9

Preliminaries

We denote the set of 4 parties as $\mathbf{P} = \{P_1, P_2, P_3, P_4\}$ and each pair is connected by pairwise-secure private channels. We next describe the primitives that are additionally required for the BoBW constructions. The remaining primitives of Garbling, Replicated Secret Sharing, Non-interactive and Equivocal Non-interactive commitment schemes are as detailed in Chapter 2.

Extractable Commitment Here, we discuss a 3-round extractable commitment protocol (C, R). We now define extractable commitments taken verbatim from [PW09]:

Definition 6. *Let (C, R) be a statistically binding commitment scheme. We say that (C, R) is an extractable commitment scheme if there exists an expected polynomial-time probabilistic oracle machine (the extractor) E that given oracle access to any PPT cheating sender C^* outputs a pair (τ, σ^*) s.t*

- *(simulation) τ is identically distributed to the view of C^* at the end of interacting with an honest receiver in the commit phase*
- *(extraction) the probability that τ is accepting and $\sigma^* = \perp$ is negligible.*
- *(binding) if $\sigma^* \neq \perp$, then it is statistically impossible to open τ to any value other than σ^*.*

Instantiation: An instantiation of an extractable commitment $(\mathsf{ExtCom}, \mathsf{ExtOpen})$ appears in Fig 9.1 with the extractor algorithm in Fig 9.2. We refer to [PW09] for details of proof (implicit in [PRS02, Ros04]) that ExtCom is an extractable commitment scheme.

Commitment phase ExtCom**:**
Let $\sigma \leftarrow \{0,1\}^m$ denote the input of S (committer / sender)

Round 1: S commits (using NICOM Com) to k pairs of strings $(v_1^0, v_1^1) \ldots (v_n^0, v_n^1)$ where

$(v_i^0, v_i^1) = (\eta_i, \sigma \oplus \eta_i)$ and $\eta_1 \dots \eta_k$ are random strings in $\{0,1\}^m$.

Round 2: R sends challenge $e = (e_1 \dots e_k)$.

Round 3: S opens the commitments to $v_1^{e_1} \dots v_k^{e_k}$. R checks if the openings are valid.

Decommitment Phase ExtOpen**:**

- S sends σ and opens the commitments to all k pairs of strings.
- R checks that all the openings are valid and also that $\sigma = v_1^0 \oplus v_1^1 = \dots v_k^0 \oplus v_k^1$.

Figure 9.1: Extractable Commitment

Fix a cheating committer C^*.

- First, simulate an execution of C^* by internally emulating an honest receiver R to obtain a transcript $\tau = \{\mathsf{extcom}_1^1, \mathsf{extcom}_2^1, \mathsf{extcom}_3^1\}$ of the commit phase. If τ is rejecting, then output $(\tau, \perp)$ and halt.
- If τ is accepting with some challenge e, then keep rewinding C^* with random challenges until we receive another accepting response from C^* with some challenge e'. If $e = e'$ then output $(\tau, \perp)$ and halt. Otherwise, extract a value σ^* from the C^*'s responses to distinct challenges e, e' (by combining the appropriate shares), and output (τ, σ^*).

Figure 9.2: Extractor Algorithm Extract

Chapter 10

Garbling Building Blocks

This chapter is primarily concerned with distributed garbling (used in both our BoBW constructions) and related building blocks. The aim is to elaborate the garbling scheme used in $\pi_{\mathsf{bobw.god}}$ to allow familiarity with the concept and notation, hence enabling smooth understanding of the techniques introduced to ensure BoBW guarantees.

10.1 Distributed Garbled Circuit [BMR90]

In multiparty setting, it is necessary for all parties to participate in the construction of garbled circuit to prevent any coalition of corrupt parties from learning information about the value being computed. We use the passively-secure scheme of [BLO16] for our BoBW protocol achieving GOD in honest majority.

In the computation of distributed garbled circuit (DGC), let $n-1$ parties be the garblers and the remaining party (P_n wlog) be the evaluator. Each wire w is associated with mask $\lambda_w \in \{0,1\}$. Each garbler P_i samples its mask share λ_w^i s.t $\oplus_{i\in[n-1]}\lambda_w^i = \lambda_w$. The technique of point and permute is used to hide the outputs of intermediate gates and λ_w acts as the permutation bit for wire w. Also, P_i chooses two keys $k_{w,0}^i$ and $k_{w,1}^i = k_{w,0}^i \oplus \Delta^i$ per wire where Δ^i is the global offset of P_i. Each wire is thus defined with a set of $n-1$ keys for 0-label and $n-1$ keys for 1-label. The keys and mask for an output wire of an XOR gate is set equal to the XOR of the keys and masks for the input wires to enable the property of free XOR [KS08]. Construction of AND gate ciphertexts, as depicted in the functionality $\mathcal{F}_{\mathsf{GC}}$ (Fig 10.1) of [BLO16], requires interaction amongst the garblers and thus is realized by all garblers running a secure MPC protocol to compute the distributed garbled circuit. Specifically, the ciphertext correspsonding to row α, β for a party P_j in $\mathcal{F}_{\mathsf{GC}}$ is realized with the use of two sets of standard Oblivious Transfer (OT) between every pair of garblers such that garbler P_j

obtains $\lambda_\gamma^j = [(\lambda_u \oplus \alpha) \cdot (\lambda_v \oplus \beta)]_j$ where λ_γ^j is the additive share of $(\lambda_u \oplus \alpha) \cdot (\lambda_v \oplus \beta)$. This is followed by the transfer of $\rho_{\alpha,\beta}^{i \to j} = \mathsf{F}^2_{k^i_{u,\alpha}, k^i_{v,\beta}}(w||j) \oplus \Delta^j \cdot \lambda_\sigma^i$ to P_j by every $P_i, i \neq j$ where $\lambda_\sigma^i = \lambda_\gamma^i \oplus \lambda_w^i$ is the additive share of $((\lambda_u \oplus \alpha) \cdot (\lambda_v \oplus \beta) \oplus \lambda_w)$. The final garbled circuit is denoted by GC which is the concatenation of $\{\mathsf{GC}^j\}_{j \in [n-1]}$ where GC^j is the fragment of the garbled circuit constructed by P_j.

Let C be the circuit, F be a PRF, κ be the security parameter. Garbler P_i has the following set of inputs:

- Global offset string Δ^i.
- Share $\lambda_w^i \in \{0,1\}$ of the mask bit λ_w for every wire w.
- Keys $k_{w,0}^i$ and $k_{w,1}^i$ for every wire w such that $k_{w,1}^i = k_{w,0}^i \oplus \Delta^i$.

Computation: The functionality computes the garbled circuit GC. Compute the keys and masks for output wire of XOR gates using free-XOR. For every AND gate with input wires u, v and output wire w, row $\alpha, \beta \in \{0,1\}$ and each garbler $P_j, j \in [n-1]$ compute:

$$c_{\alpha,\beta}^j = \Big(\bigoplus_{i \in [n-1]} \mathsf{F}^2_{k^i_{u,\alpha}, k^i_{v,\beta}}(w||j) \Big) \oplus k_{w,0}^j \oplus \big(\Delta^j \cdot ((\lambda_u \oplus \alpha) \cdot (\lambda_v \oplus \beta) \oplus \lambda_w) \big)$$

Output: Output $\mathsf{GC}^j = \{c_{\alpha,\beta}^j\}_{\forall \text{AND gates}}$ to P_j.

Figure 10.1: Functionality $\mathcal{F}_{\mathsf{GC}}$

Evaluation. Evaluation of DGC is performed on masked inputs. Specifically, the evaluator P_n gets $\{k^i_{u,m_u}, k^i_{v,m_v}\}_{i \in [n-1]}$ and masked inputs (m_u, m_v) where $m_u = x_u \oplus \lambda_u, m_v = x_v \oplus \lambda_v$ for gate g with input wires u, v (x_u, x_v are the actual inputs on u, v respectively) and output wire w. If g is an XOR gate, then, P_n sets $k^i_{w,m_w} = k^i_{u,m_u} \oplus k^i_{v,m_v}$ for each i and $m_w = m_u \oplus m_v$ where m_w is the masked output on w. If g is an AND gate, then P_n decrypts the row (m_u, m_v) of the corresponding garbled gate of GC^i to obtain $m_w = x_u x_v \oplus \lambda_w$ and k^i_{w,m_w} for $i \in [n-1]$. For output wire w, the output mask λ_w is revealed to compute the output $x_w = m_w \oplus \lambda_w$. We use $\mathbf{Y}$ (encoded output) to denote the set of $n-1$ keys on all output wires obtained on evaluation of GC.

For our BoBW protocol achieving GOD in honest majority, we modify the functionality $\mathcal{F}_{\mathsf{GC}}$ to segregate the messages computed by each garbler such that the ciphertext $c_{\alpha,\beta}^j$ of a party P_j is formulated using only the data present with P_j after the OT step and no transfer of $\rho_{\alpha,\beta}^{i \to j}$ is done to P_j by any $P_i, i \neq j$ (instead $\rho_{\alpha,\beta}^{i \to j}$ is directly sent to the evaluator as part of P_i's ciphertext for the evaluator to do the needful). This is done to facilitate the correct identification of a corrupt garbler P_j in case of a faulty GC construction once the robustness

of OT step is ensured. In $\mathcal{F}_{\mathsf{GC}}$, it is not correct to implicate P_j in case of a faulty GC^j received from P_j because P_j also comprises of data sent by $P_i, i \neq j$ which might have caused GC^j to be incorrect. For the modified functionality $\mathcal{F}_{\mathsf{GCMod}}$, we show only the modification required in the computation step of $\mathcal{F}_{\mathsf{GC}}$ in Fig 10.2.

Computation: For every AND gate with input wires u, v and output wire w, every $\alpha, \beta \in \{0,1\}$ and each garbler $P_j, j \in [n-1]$ compute:

$$c^j_{\alpha,\beta} = \left(\mathsf{F}^2_{k^j_{u,\alpha},k^j_{v,\beta}}(w||j) \oplus k^j_{w,0} \oplus \left(\Delta^j \cdot \lambda^j_\sigma\right)\right)||\left(||_{i\neq j}\rho^{j\to i}_{\alpha,\beta}\right)$$

where $\oplus_{j\in[n-1]}\lambda^j_\sigma = (\lambda_u \oplus \alpha)\cdot(\lambda_v \oplus \beta) \oplus \lambda_\gamma$ and $\rho^{j\to i}_{\alpha,\beta} = \mathsf{F}^2_{k^j_{u,\alpha},k^j_{v,\beta}}(w||i) \oplus \Delta^i \cdot \lambda^j_\sigma$

Output: Output $\mathsf{GC}^j = \{c^j_{\alpha,\beta}\}_{\forall \text{AND gates}}$ to P_j.

Figure 10.2: Modified Functionality $\mathcal{F}_{\mathsf{GCMod}}$ of $\mathcal{F}_{\mathsf{GC}}$

For our BoBW construction achieving fairness in honest majority, we use the state of the art maliciously secure garbling scheme of [WRK17] which is discussed in detail in Chapter 12.

10.2 Seed-distribution

The starting point of the BoBW protocol achieiving GOD in honest majority, $\pi_{\mathsf{bobw.god}}$ is a semi-honest distributed garbling scheme (as per functionality $\mathcal{F}_{\mathsf{GCMod}}$) with $\{P_1, P_2, P_3\}$ as garblers and P_4 as evaluator. The final garbled circuit (DGC) is denoted as $\mathsf{GC} = \mathsf{GC}^1||\mathsf{GC}^2||\mathsf{GC}^3$ where GC^g ($g \in [3]$) denotes the g^{th} fragment of GC. To tackle the malicious adversary while still relying on passively secure DG, a mechanism is needed to ensure correctness of the GC. We adopt the technique of seed-distribution (SD) used in honest-majority 5PC construction of [CGMV17] and scale it for 4PC to ensure correctness of GC in the face of dishonest minority i.e. 1 active corruption. SD enables a pair of parties to construct each fragment of GC and correctness of that fragment is verified by simply checking the equality of the copies. For this, we assume that all the randomness used to construct GC fragment GC^g by the designated garbler (say P_i) is derived from seed s_g and s_g is given to another garbler (say P_j). Now, both P_i and P_j construct GC^g and send to the evaluator. This strategy suffices for 1 active corruption since one of the seed-owners is honest and is guaranteed to construct the GC fragment honestly.

Our seed-distribution works as follows: Three seeds $\mathsf{s}_1, \mathsf{s}_2, \mathsf{s}_3$ are distributed amongst the garblers P_1, P_2, P_3 such that party P_g holds all but seed s_g. As a result, the fragment GC^1 constructed from seed s_1 (analogously GC^2 and GC^3) is sent by two parties P_2, P_3 who hold seed s_1. Additionally, this technique also maintains input privacy for colluding parties (dishonest

majority case in BoBW GOD construction: 2 corruptions) since, (a) for 2 corrupt garblers, all seeds are known to the adversary but the evaluator is guaranteed to be honest; (b) a colluding garbler and the evaluator lack the knowledge of *one* seed, hence the secrets remain hidden from the adversary. We denote by $\mathcal{S}_g$, the indices of the seeds held by party P_g as well as the indices of the parties who hold seed s_g i.e. $\mathcal{S}_1 = \{2,3\}, \mathcal{S}_2 = \{1,3\}, \mathcal{S}_3 = \{1,2\}$.

Notation: $\mathcal{S}_1 = \{2,3\}$, $\mathcal{S}_2 = \{1,3\}$, $\mathcal{S}_3 = \{1,2\}$.
Output: Party $P_g, g \in [3]$ outputs seed $\mathsf{s}_i, i \in \mathcal{S}_g$.

Seed-setup:
- P_1 samples a random seed s_2 and runs the routine ExtCom with P_3 as receiver. The broadcast-only transcript of ExtCom (hence, the commitment) is available to the remaining parties too.
- P_1 runs the routine ExtOpen and sends opening $o[\mathsf{s}]_2$ privately to P_3. If the opening is invalid, P_3 aborts. Else, P_3 computes s_2 using opening $o[\mathsf{s}]_2$.
- Similar steps are done by P_2 for seed s_3 and P_3 for seed s_1.

Figure 10.3: Seed-distribution π_{seedDist}

For our purposes, SD is done by broadcasting commitment on each seed and sending the opening to only the designated seed-owner. This is done for a purpose elaborated in Chapter 11. Also, extractable commitments are used for commitment to the seeds to handle a technicality arising in the proof.

10.3 Attested Oblivious Transfer

Attested Oblivious Transfer (AOT) is an inexpensive symmetric-key variant of OT between a sender and a receiver with additional help from a third party called attester to ensure correctness of OT. The primitive of AOT was introduced in the 5PC construction of [CGMV17] which replaced the semi-honest OTs required in [BLO16] with AOTs. We recall the functionality $\mathcal{F}_{\mathsf{aot}}$ in Fig 10.4.

P_s, P_r act as sender and receiver respectively, P_a is the attester.
- On input message (`Sen`, m_0, m_1) from P_s, record (m_0, m_1) and send (`Sen`, m_0, m_1) to P_a and `Sen` to the adversary.
- On input message (`Rec`, b) from P_r, where $b \in \{0,1\}$, record b and send (`Rec`, b) to P_a and `Rec` to the adversary.
- On input message (`Att`, m_0^a, m_1^a, b^a) from P_a, if (`Sen`, `sid`, $*$, $*$) and (`Rec`, $*$) have not been recorded, ignore this message; otherwise, record (m_0^a, m_1^a, b^a) and send `Att` to the adversary.
- On input message `Output` from the adversary, if $(m_0, m_1, b) \neq (m_0^a, m_1^a, b^a)$, send (`Output`, $\perp$) to

P_r; else send $(\texttt{Output}, m_b)$ to P_r.

- On input message `abort` from the adversary, send $(\texttt{Output}, \perp)$ to P_r.

Figure 10.4: Functionality $\mathcal{F}_{\mathsf{aot}}(P_s, P_r, P_a)$

In our BoBW GOD construction, due to the used of seed-distribution, standard OTs used in the $\mathcal{F}_{\mathsf{GC}}$ [BLO16] and $\mathcal{F}_{\mathsf{GCMod}}$ are replaced with AOTs. Specifically, SD ensures that for every OT that is supposed to run between a sender P_s and a receiver P_r, there exists an attester P_a who possesses the seeds (hence, the OT inputs) of both P_s and P_r. For instance when the sender's messages are derived from seed s_2 and receiver's choice bit is derived from seed s_1, P_3 is set to be the attester since he knows both seeds s_1 and s_2. P_1 (holding s_2) and P_2 (holding s_1) act as sender and receiver respectively.

For the BoBW GOD construction, to ensure BoBW guarantees, a modified version of the AOT fuctionality $\mathcal{F}_{\mathsf{aot}}$ is used which is elaborated in Chapter 11.

Chapter 11

GOD in Best-of-Both-Worlds Setting

In this chapter, we present a BoBW protocol $\pi_{\mathsf{bobw.god}}$ for 4 parties that promises: a) GOD against 1 active corruption ($t = 1$) and b) unanimous abort (UA) against 2 active corruptions ($s = 2$). The corruption thresholds are optimal, adhering to the feasibility constraint of $t+s < n$ [IKLP06].

At a high-level, we achieve our goal of attaining 4PC BoBW security by first identifying and eliminating a corrupt party and then engaging the remaining 3 parties in a 3PC UA protocol. Specifically, when a corrupt party is eliminated, the remaining 3 parties may either have *no* corrupt party (in which case the 3PC will be robust and our 4PC achieves GOD as required) or *one* corrupt party (in which case the 3PC achieves UA, as demanded by our 4PC against two corruptions).

11.1 The Construction

The high-level framework involves a passively-secure distributed garbling (DG) scheme as in $\mathcal{F}_{\mathsf{GCMod}}$ (Fig 10.2), combined with a bunch of additional techniques optimized for our setting and security expectations. We structure the protocol with $\{P_1, P_2, P_3\}$ as the three garblers and P_4 as the evaluator. The novelty of our construction lies in tackling a horde of challenges (and maintaining efficiency) that surface while putting together these techniques. The protocol starts with a *robust seed-distribution* (SD) and *input-commit* routines. We segregate the rest of construction in the following phases, each of which is geared with tools that enable the honest parties to either unanimously identify a corrupt party (and then, run a 3PC instance) or identify the presence of 2 corruptions (and then, abort).

Input-Insensitive Phase This includes the transfer of messages for which it is safe to reveal the underlying randomness in case of a misbehaviour. It primarily involves distributed garbling

as per functionality $\mathcal{F}_{\mathsf{GCMod}}$ (Fig 10.2) enabled with the technique of seed-distribution (SD) as explained in Chapter 10. SD involves distribution of seeds $\{\mathsf{s}_1, \mathsf{s}_2, \mathsf{s}_3\}$ amongst the garblers $\{P_1, P_2, P_3\}$ as elaborated in Chapter 10, where a garbler P_g knows all but seed s_g. The formal SD protocol π_{seedDist} appears in Fig 10.3. For our purposes, we perform the seed distribution by broadcasting commitment on each seed and opening the commitment to only the relevant designated seed-owner. This is done to ease the identification of corrupt party in case of any wrongdoing. To elaborate, SD enables two garblers to be able to construct (and broadcast) each GC fragment. In case of mismatching copies of a fragment sent by the two seed-owners, they are required to publicly reveal (broadcast) the opening of the corresponding seed so that each honest party can compute the seed (by using the opening and the broadcast-only commmitment routine ExtCom). The seed is then used to locally construct the GC fragment to pin-point the actual corrupt sender. The modified DGC functionality $\mathcal{F}_{\mathsf{GCMod}}$ is crucial in pin-pointing the corrupt party in case of mismatch since, once the AOT step is made robust, the ciphertext generated wrt each seed involves data only from local computation using the respective seed. With public revelation of seeds in case of mismatch, the GC fragment as computed by $\mathcal{F}_{\mathsf{GCMod}}$ can be computed locally and the wrongdoer can be correctly and unanimously identified.

Further, we modify the AOT functionality $\mathcal{F}_{\mathsf{aot}}$ to incorporate the technique of opening of seeds if the broadcasted commitments of the sender and the attester mismatch (to identify *one* active corruption) and the resulting protocol $\pi_{\mathsf{aot.bobw}}$ appears in Fig 11.1.

P_s, P_r denote the sender and receiver respectively. P_a denotes the attester and P_h denotes the helper.

Notations: Con_k, Corr_k respectively denote conflict and corrupt set of P_k. $\mathbb{C}^3$ is the 3PC committee with at most 1 corruption.

Input and Output: P_s inputs m_0, m_1, P_r inputs choice bit b. P_r outputs $m_b/\mathbb{C}^3$. All other parties output $\perp/\mathbb{C}^3$.

Primitives: A secure NICOM $(\mathsf{Com}, \mathsf{Open})$.

Round 1:

- P_s samples pp and random values $r_0, r_1 \leftarrow \{0,1\}^\kappa$ (derived from $\mathsf{s}_g, g \in \mathcal{S}_s \cap \mathcal{S}_a$) to compute $(\mathsf{c}_0, \mathsf{o}_0) \leftarrow \mathsf{Com}(\mathsf{pp}, m_0)$ and $(\mathsf{c}_1, \mathsf{o}_1) \leftarrow \mathsf{Com}(\mathsf{pp}, m_1)$. P_s broadcasts $(\mathsf{pp}, \mathsf{c}_0, \mathsf{c}_1)$.
- P_a, who knows (r_0, r_1) (derived from s_g), also computes $(\mathsf{c}'_0, \mathsf{o}'_0) \leftarrow \mathsf{Com}(\mathsf{pp}, m_0)$ and $(\mathsf{c}'_1, \mathsf{o}'_1) \leftarrow \mathsf{Com}(\mathsf{pp}, m_1)$. P_a broadcasts $(\mathsf{c}'_0, \mathsf{c}'_1)$. P_a also broadcasts $\mathsf{o}'_b \oplus \mathsf{ran}_{ar}$ where ran_{ar} is the randomness agreed between P_a and P_r in $\pi_{\mathsf{mask.bobw}}$ (Fig 11.2).

Local computation:

- If either $\mathsf{c}_0 \neq \mathsf{c}'_0$ or $\mathsf{c}_1 \neq \mathsf{c}'_1$: P_s and P_a broadcast $o[\mathsf{s}]_g$ (opening of the seed s_g common between P_s,

P_a) which is used by each party $P_i \in \mathbf{P}$ to compute s_g and verify the correctness of commitments sent by P_a, P_s to conclude the corrupt party and update Corr_i. P_i outputs $\mathbb{C}^3 = [4] \setminus \mathsf{Corr}_i$.

- Else, P_r unmasks ran_{ar} and uses o'_b (if valid) to compute $m_b \leftarrow \mathsf{Open}(\mathsf{c}'_b, \mathsf{o}'_b)$. P_r outputs m_b and others output $\perp$. If o'_b is invalid, P_r broadcasts $o[\mathsf{ran}]_{ar}$ for each $P_i \in \mathbf{P}$ to compute $\mathsf{ran}_{ar} \leftarrow \mathsf{Open}(c[\mathsf{ran}]_{ar}, o[\mathsf{ran}]_{ar})$, obtain o'_b to conclude the identity of the corrupt party and update Corr_i accordingly. Each party outputs $\mathbb{C}^3 = [4] \setminus \mathsf{Corr}_i$.

Figure 11.1: Modified Attested OT $\pi_{\mathsf{aot.bobw}}(P_s, P_r, P_a, P_h)$

Apart from GC communication, this phase involves the transfer of some input-insensitive mask-shares which is discussed next. As explained in Chapter 10, each wire of the circuit is associated with a mask. Besides, the evaluation of GC is done using the keys for masked inputs and the GC outputs masked values for the output wires. To obtain clear outputs, the mask values need to be provided to each party to enable unmasking. The mask-shares on output wires are broadcast to ensure easy identification of the corrupt party using the above technique of opening the underlying seeds in case of mismatch. Also, since the evaluation is done on masked inputs, the input-wire owner must know the mask on the input wire. A garbler acting as wire owner is missing one seed (hence, one share) while the evaluator acting as wire owner is missing all shares. Each garbler is asked to broadcast the shares corresponding to the seeds the wire owner is missing. And a mismatch is handled as done for the output mask shares.

Cut-and-Choose Note that SD ensures security of the DGC against only *one* actively corrupt garbler as there is always an honest party to send the correct GC fragment for verification. However, in case of *two* active corruptions among garblers (say P_1, P_2), correctness of the GC can no longer be ensured with SD alone since the two corrupt garblers have the sole ownership of one seed (s_3) and the resulting DGC fragment (GC^3). They could both send matching faulty copies of GC^3 leading to no suspicion. This could further cause privacy breach of the honest parties' input due to arbitrary function being computed by GC. To tackle this, we rely on the technique of *cut-and-choose* [Lin13] to verify the correctness of GC and its related information. This is reminiscent of the strategy adopted by 3PC of [CKMZ14] to tackle 2 corrupt garblers. Note that, any failure in the *check* circuits can be attributed to the presence of *two* corruptions (in which case the security expectation is UA), hence the honest parties abort. Any wrong-doing in $\pi_{\mathsf{aot.bobw}}$ in case of two corruptions (particularly when both sender and attester broadcast incorrect commitments) goes undetected at the time of GC computation but is caught in the cut-and-choose phase as the AOT transcript is public due to the use of broadcast messages in $\pi_{\mathsf{aot.bobw}}$. Cut-and-choose brings along concerns for input privacy if any input sensitive data has already been exchanged in the protocol as the underlying seeds are revealed to open the

check circuits. Hence, we structure the protocol to have the input-sensitive communication after cut-and-choose step is complete.

For the purpose of choosing a challenge string in cut-and-choose, each party P_i commits to a random string ρ_i of length s at the onset of the protocol, while the openings are revealed only in this phase. The final challenge string is the XOR of each ρ_i. Note that cut-and-choose involves the problem of input inconsistency as pointed out in [Lin16, CKMZ14]: a corrupt P_1, P_2 pair can send inconsistent encoding labels across different *evaluation* circuits for the input wires (w.r.t. shares owned by P_1, P_2) enforcing P_4 to evaluate on multiple inputs. We tackle this well-known attack using the Diffie-Hellman pseudorandom synthesizer trick [LP11, MF06] by adapting it to our setting in a straight-forward manner.

Input-Sensitive Phase The above techniques to identify misbehaviour that rely on broadcast can be applied only to input-insensitive data because of threat to input privacy when the seeds are opened. For most private input-dependent messages, π_{seedDist} or $\pi_{\mathsf{ICom.bobw}}$ ensure that there exist 2 parties who can send the message privately to the receiver. This comes with many challenges: (i) The receiver must be able to identify the corrupt sender (out of the two), (ii) He must be able to convince the remaining honest parties that the identified sender is corrupt. To resolve (i), we rely on the technique of **commit publicly, open privately** where both senders broadcast the commitments to the private message (which are compared and a mismatch is tackled as in *input-insensitive phase*) and the openings are sent in private to the designated receiver. In case of invalid openings received privately, the receiver (say P_r) has identified the sender(s) to be corrupt and can raise a public conflict against the corrupt sender (say P_s). Note that this does not suffice for the remaining honest parties to identify P_s to be corrupt as a possibly corrupt P_r could raise a false conflict against an honest P_s leading to the problem in case (ii). We resolve this by introducing the technique of **oblivious broadcast**. To elaborate, we establish a pre-agreed *random mask* between each pair of parties at the onset of the protocol and enforce the senders to broadcast the opening XORed with the random mask pre-agreed between the sender and the receiver (so that the broadcast is meaningful only to the receiver). This mask is agreed upon by a process similar to π_{seedDist} where the commitments to the mask are broadcast and openings are sent in private. The formal random-mask distribution setup $\pi_{\mathsf{mask.bobw}}$ is given in Fig 11.2. This technique ensures that the masked opening is always received by the receiver. Now if the receiver finds the opening (after unmasking) to be invalid, she reveals the random mask (particularly, its opening) for everyone to publicly identify the wrongdoer. Note that this leads to each party learning the private message which is safe, since conflict occurs only when the adversary is involved and is already aware of the underlying message.

Output: Party $P_i, i \in [4]$ outputs ran_{ij} $(j > i)$ and ran_{ji} $(j < i)$.

Random-mask setup: P_i, P_j for $i, j \in [4], i < j$ do as follows:

- P_i samples pp_i and a random-mask value ran_{ij} to compute $(c[\mathsf{ran}]_{ij}, o[\mathsf{ran}]_{ij}) \leftarrow \mathsf{Com}(\mathsf{pp}_i, \mathsf{ran}_{ij})$.
- P_i broadcasts $(\mathsf{pp}_i, c[\mathsf{ran}]_{ij})$; sends $o[\mathsf{ran}]_{ij}$ privately to P_j.
- P_j computes $\mathsf{ran}_{ij} \leftarrow \mathsf{Open}(\mathsf{pp}_i, c[\mathsf{ran}]_{ij}, o[\mathsf{ran}]_{ij})$. If the opening is invalid, P_j aborts.

Figure 11.2: Random-mask distribution routine $\pi_{\mathsf{mask.bobw}}$

For the transfer of masked input bits, we combine the techniques of *commit publicly, open privately* and *oblivious broadcast* where the commitments (generated using randomness other than derived from seeds for privacy concerns) is broadcast in the input-insensitive phase while the (oblivious) opening is broadcast in this phase. Any inconsistency is handled by asking the senders to reveal the related seeds / random-masks. For the transfer of encoding labels wrt masked inputs to P_4, we follow the standard technique of [MRZ15, CGMV17] to have each party broadcast the commitments to both labels in input-insensitive phase (where the misbehaviour is handled as explained). We came across various challenges in the transfer of relevant openings while maintaining BoBW security guarantees *and* input privacy which we addressed as follows:

(I) *For shares owned by two garblers (say x_{12}):* Both the share owners (P_1, P_2) send key-openings corresponding to masked input b_{12} wrt the seeds that they own. If an opening is found to be invalid, P_4 raises a conflict with the garbler (who sent the invalid opening) and broadcasts b_{12}. The remaining 2 garblers broadcast key-openings wrt the seeds they own. If a broadcasted opening is again found to be invalid, then the honest parties conclude that they are in the setting of $s = 2$ and hence abort. Note that broadcasting b_{12} is safe because either it is a share that already belongs to the adversary (when one of P_1, P_2 is corrupt) or the adversary doesn't have enough seeds to know the underlying x_{12} (when both P_1, P_2 are honest).

(II) *For shares owned by P_4 and a garbler (say P_1) i.e. x_{14}:* P_1 sends openings corresponding to masked input b_{14} wrt seeds s_2 and s_3. Case (I) is followed in case of invalid opening. For opening wrt seed s_1 not held by P_1 (possessed by both P_2 and P_3), P_1 runs a 1-out-of-2 malicious OT acting as receiver with choice bit b_{14} and P_2 as sender with inputs as keys while P_4 runs a similar OT acting as receiver with P_3 as sender. If P_4 does not receive a valid opening from her OT with P_3 she waits for a valid opening from P_1 (who sends the opening he received as his OT output privately to P_4 only if it is valid). If still not received a valid opening, P_4 concludes the presence of two corruptions (P_3 and P_1/P_2) and broadcasts `abort`. An honest P_1 who did not receive a valid opening from her OT with P_2 also aborts

on receiving `abort` from P_4 as P_1 concludes the presence of two corruptions, P_2 and P_3/P_4. Lastly, P_1 also broadcasts `abort` in such a case to bring P_2, P_3 on the same page.

Also, the *selective-failure* attack in which a corrupt garbler can give one valid and one invalid label as input to the OT in order to learn the receiver's input by checking whether the evaluation succeeded or not is handled by directly using the *XOR-tree* technique [LP07]. Finally, if all valid labels are received, P_4 evaluates the GC and broadcasts $\mathbf{Y}$ to enable all parties to compute the output. In case a corrupt P_4 broadcasts an invalid (or no) $\mathbf{Y}$, each party unanimously identifies P_4 to be corrupt and switch to a 3PC without P_4.

Lastly, we tackle the input consistency issue that arises when the corrupt P_4 aborts after obtaining the output herself causing the remaining parties to rely on the 3PC-instance for the output. To prevent the adversary from obtaining multiple evaluations of f, the 3PC should use the same inputs as in the 4PC. This is ensured using the robust $\pi_{\mathsf{ICom.bobw}}$, where the commitments on input shares are broadcast and agreed upon while the technique of *oblivious broadcast* is used to reveal openings. The formal protocol of $\pi_{\mathsf{ICom.bobw}}$ is presented in Fig 11.3. To elaborate, since the commitments on all input shares are already agreed upon, the circuit computed for the 3PC instance has the commitments of the input-shares hardcoded and takes the openings (w.r.t. all shares) from the participating parties in the 3PC-instance as input. The circuit checks the validity of openings and computes the input-shares. Further, a potential corrupt party in the chosen 3PC cannot give a different valid opening (due to the strong binding property of the NICOM) to obtain multiple outputs. An invalid opening (leading to abort) can be given in the 3PC instance only by a corrupt party amongst the participants which is an indication of presence of 2 corruptions and thus allowing the circuit to output $\perp$ in such case is an acceptable output. The 3PC instance appears in Fig 11.4.

Input: P_i has input x_i.

Output: Every $P_j, j \neq i$ outputs $(\mathsf{pp}_i, c[\mathsf{in}]_{ij}, o[\mathsf{in}]_{ij})$ or $\mathbb{C}^3$.

Primitives: A secure NICOM $(\mathsf{Com}, \mathsf{Open})$.

Computation: P_i splits her input as $x_i = \oplus_{j\in[4]\setminus\{i\}} x_{ij}$ and computes commitments as $(c[\mathsf{in}]_{ij}, o[\mathsf{in}]_{ij}) \leftarrow \mathsf{Com}(\mathsf{pp}_i, x_{ij})$. P_i broadcasts $(\mathsf{pp}_i, c[\mathsf{in}]_{ij})$ and $o[\mathsf{in}]_{ij} \oplus \mathsf{ran}_{ij}$ where ran_{ij} is shared randomness between P_i, P_j agreed in $\pi_{\mathsf{mask.bobw}}$ (Fig 11.2).

Local Computation by P_j: Unmask ran_{ij} to obtain $o[\mathsf{in}]_{ij}$ and compute $x_{ij} = \mathsf{Open}(\mathsf{pp}_i, c[\mathsf{in}]_{ij}, o[\mathsf{in}]_{ij})$. If $o[\mathsf{in}]_{ij}$ is invalid, broadcast $o[\mathsf{ran}]_{ij}$. Each $P_k \in \mathbf{P}$ uses $o[\mathsf{ran}]_{ij}$ (if valid) to verify the validity of $o[\mathsf{in}]_{ij}$ and determines the corrupt party out of P_i, P_j to set Corr_k. Run $\mathsf{3PC}$ with $\mathbb{C}^3 = \mathbf{P} \setminus \mathsf{Corr}_k$.

Figure 11.3: Input-commit routine $\pi_{\mathsf{ICom.bobw}i}$

Notation: Let $P_\alpha, P_\beta, P_\gamma$ be the participating parties and P_δ be the corrupt party identified.

Inputs: Party P_α (similarly P_β and P_γ) has inputs $(o[\mathsf{in}]_{\alpha i}, o[\mathsf{in}]_{\delta\alpha})_{i\in[4]\setminus\{\alpha\}}$.

Common Inputs: The circuit C that takes the openings $o[\mathsf{in}]_{ij}$ for $i \in [4], j \in [4] \setminus \{i\}$ as inputs, checks whether $o[\mathsf{in}]_{ij}$ is valid w.r.t. $c[\mathsf{in}]_{ij}$, computes x_{ij} and $x_i = \oplus_{j\in[4]\setminus\{i\}} x_{ij}$ to compute $f(x_1, x_2, x_3, x_4)$.

Output: $P_i \in \mathbf{P}$ outputs $y = f(x_1, x_2, x_3, x_4)$ or $\perp$.

Computation: Run 3PC of [MRZ15] with P_α and P_β as the garblers and P_γ as the evaluator except the following change: P_γ broadcasts $\mathbf{Y}$ instead of sending $\mathbf{Y}$ privately in round 3 of [MRZ15] to ensure unanimous abort.

Figure 11.4: Three-party instance $\mathsf{3PC}()$

Although it may appear that, use of a maliciously secure garbling scheme to begin with could eliminate the need of cut-and-choose and ease several other challenges, however, the best-known maliciously secure garbling schemes [WRK17] lack the property of *identifiability* in case of corrupt behaviour and tools necessary to include such a property appear to be much more expensive than directly using cut-and-choose. This completes the intuition. The formal protocol appears in Fig 11.5.

Input and Output: Each party $P_i \in \mathbf{P}$ has x_i. Each party outputs $y = f(x_1, x_2, x_3, x_4)$.

Common Inputs: The circuit C that takes additive shares x_{ij} of x_i for $i \in [4], j \in [4] \setminus \{i\}$ as inputs and computes $f(x_1, x_2, x_3, x_4)$, each input, their shares and output are from $\{0,1\}$ (instead of $\{0,1\}^\ell$ for simplicity). s is the statistical security parameter.

Primitives: A secure NICOM $(\mathsf{Com}, \mathsf{Open})$, Extractable Commitment $(\mathsf{ExtCom}, \mathsf{ExtOpen})$, Oblivious Transfer (OT) and collision resistant hash H, π_{seedDist} (Fig 10.3), $\pi_{\mathsf{mask.bobw}}$ (Fig 11.2), $\pi_{\mathsf{ICom.bobw}}$ (Fig 11.3).

Seed and mask distribution: For each $\mathsf{itr} \in [s]$, parties P_1, P_2, P_3 run π_{seedDist}. Parties $P_i \in \mathbf{P}$ run $\pi_{\mathsf{mask.bobw}}$.

Input-sharing: Run $\pi_{\mathsf{ICom.bobw}i}$ for each $P_i \in \mathbf{P}$.

Cut-and-choose challenge string: $P_i, i \in [4]$ samples random $\rho_i \leftarrow \{0,1\}^s$ and commits to ρ_i with P_{i+1} as receiver using the extractable commitment routine ExtCom.

(I) Input-insensitive Phase: Run this phase for each $\mathsf{itr} \in [s]$:
- $P_g, g \in [3]$ broadcasts $\lambda_w^h, h \in \mathcal{S}_g$ for *output* wire w.
- For every *input* wire w w.r.t. x_w held by two garblers: for each P_g (owning x_w), garbler $P_j, j \neq g$

broadcasts λ_w^g. (If P_4 holds x_w, then garbler $P_j, j \in [3]$ broadcasts $\lambda_w^l, l \in \mathcal{S}_j$).

- Let $k_{w,0}^h$ and $k_{w,1}^h$ denote the two keys derived from seed s_h for input wire w. $P_g, g \in [3]$ computes commitments for $h \in \mathcal{S}_g$ and $b \in \{0,1\}$ as: $(c[\mathsf{k}]_{w,b}^h, o[\mathsf{k}]_{w,b}^h) \leftarrow \mathsf{Com}(\mathsf{pp}^h, k_{w,b}^h)$ and broadcasts $\{c[\mathsf{k}]_{w,b}^h\}$.

If different copies of λ_w^g (for *output* wire w) or λ_w^g or $\{c[\mathsf{k}]_{w,b}^g\}$ (for some *input* wire w, $b \in \{0,1\}$) are broadcast for some $g \in [3]$ by (P_α, P_β) with $\alpha, \beta \in \mathcal{S}_g$: P_α, P_β broadcast opening of seed s_g ($o[\mathsf{s}]_g$). Each party uses valid $o[\mathsf{s}]_g$ (if any) to compute s_g and verifies the correctness of broadcast by P_α, P_β to determine the corrupt party (say P_α). Run $\mathsf{3PC}$ with $\mathbb{C}^3 = \mathbf{P} \setminus \{P_\alpha\}$. (Abort if two corrupt parties are identified).

In case of no mismatch, owner P_j of the input wire w computes $\lambda_w = \oplus_{i \in [3]} \lambda_w^i$ and sets $b_w = x_w \oplus \lambda_w$.

Commitment on masked inputs: For input wire w, with masked input b_w held by parties P_i, P_j, the parties compute $(c[b]_w, o[b]_w) = \mathsf{Com}(\mathsf{pp}^{ij}, b_w)$ (using randomness derived from shared ran_{ij}). P_i, P_j broadcast $(\mathsf{pp}^{ij}, c[b]_w)$. If the copies mismatch, P_i, P_j broadcast $o[\mathsf{ran}]_{ij}$, $\{o[\mathsf{s}]_l\}_{l \in [3]}$ and $o[\mathsf{in}]_{ij}$ (w.r.t. b_w) to allow every party to compute and verify the commitments of P_i, P_j to determine the corrupt party (say P_i). Run $\mathsf{3PC}$ with $\mathbb{C}^3 = \mathbf{P} \setminus \{P_i\}$. (Abort if two corrupt parties are identified).

Oblivious Transfer: For input wire w of x_w held by P_4, $P_g, g \in [3]$:

- P_g sends $o[\mathsf{k}]_{w,b_w}^h, h \in \mathcal{S}_g$ to P_4 for P_4 to compute k_{w,b_w}^h.
- If $o[\mathsf{k}]_{w,b_w}^h$ is invalid, P_4 broadcasts $o[b]_w$ and (Con, P_4, P_g). Garblers $P_j, j \neq g$ broadcast $o[\mathsf{k}]_{w,b_w}^h$ where $h \in \mathcal{S}_j$. If any opening is still invalid (can be checked by all), honest parties conclude the presence of two corrupt parties and abort. Else, P_4 uses valid openings to obtain $k_{w,b_w}^j, j \in [3]$.
- To obtain $o[\mathsf{k}]_{w,b_w}^g$: Let $\{i,j\} = [3] \setminus \{g\}, i < j$. P_g runs OT as receiver (choice bit b_w) and P_i as sender (inputs $o[\mathsf{k}]_{w,0}^g, o[\mathsf{k}]_{w,1}^g$). If the received $o[\mathsf{k}]_{w,b_w}^g$ is a valid opening, P_g forwards $o[\mathsf{k}]_{w,b_w}^g$ to P_4. Similarly, P_4 runs OT as receiver (choice bit b_w) and P_j as sender (inputs $o[\mathsf{k}]_{w,0}^g, o[\mathsf{k}]_{w,1}^g$). If valid $o[\mathsf{k}]_{w,b_w}^g$ obtained from OT, P_4 uses $o[\mathsf{k}]_{w,b_w}^g$ to compute k_{w,b_w}^g. Else if, received valid $o[\mathsf{k}]_{w,b_w}^g$ from P_g, P_4 computes k_{w,b_w}^g. Else, P_4 broadcasts `abort`.
- P_1 broadcasts `abort` if: P_1 receives invalid opening from OT *and* P_4 broadcasts `abort`. Honest parties abort if both P_1 *and* P_4 broadcast `abort`.

Transfer of GC:

- Run DG to realize $\mathcal{F}_{\mathsf{GCMod}}$ enabled with SD where $\pi_{\mathsf{aot.bobw}}$ (Fig 11.1) is used as a means to achieve OT. $\mathsf{3PC}$ is run with $\mathbb{C}^3$ when any instance of $\pi_{\mathsf{aot.bobw}}$ returns $\mathbb{C}^3$.
- Each $P_g, g \in [3]$ broadcasts $\{\mathsf{GC}^h\}, h \in \mathcal{S}_g$. If different copies of GC^g are broadcast for some $g \in [3]$ by (P_α, P_β) with $\alpha, \beta \in \mathcal{S}_g$, P_α, P_β broadcast $o[\mathsf{s}]_g$. Each party uses valid $o[\mathsf{s}]_g$ (if any) to compute s_g and verifies the correctness of GC broadcast by P_α, P_β to determine the corrupt party (say P_α). Run $\mathsf{3PC}$ with $\mathbb{C}^3 = \mathbf{P} \setminus \{P_\alpha\}$. (Abort if two corrupt parties are identified).

(II) Cut-and-choose Phase:

- Let the values broadcasted corresponding to seed s_g in the *input-insensitive* phase in iteration itr be denoted by $\mathbf{B}[g]_{\mathsf{itr}}$.
- $P_i, i \in [4]$ broadcasts the opening of ρ_i for $P_k \in \mathbf{P}$ to compute ρ_i. P_k constructs $\rho = \oplus_{i\in[4]}\rho_i$.
- Let $\mathsf{CC} = \{k : \rho^k = 1\}$ denote the check circuits (ρ^k denotes the k^{th} bit of ρ) and $\mathsf{EC} = [s] \setminus \mathsf{CC}$ denote the evaluation circuits.
- For $\mathsf{itr} \in \mathsf{CC}$, the following is done:
  - $P_g, g \in [3]$ broadcasts openings $o[\mathsf{s}]_{(g \mod 3)+1}$ and $o[\mathsf{ran}]_{gh}$ for $(h > g)$ (as per π_{seedDist}, $\pi_{\mathsf{mask.bobw}}$ respectively).
  - If P_g broadcasts an invalid opening, run $\mathsf{3PC}$ with $\mathbb{C}^3 = \mathbf{P} \setminus \{P_g\}$. Else, obtain $\mathsf{s}_{(g \mod 3)+1}$ and ran_{gh} for $h > g$.
  - Each party checks if $\mathbf{B}[g]_{\mathsf{itr}}$ sent by $P_h, h \in [3] \setminus \{g\}$ is consistent with that computed using s_g and ran_{gh} for $h > g$. If not, aborts.

(III) Input-sensitive Phase: Run this phase for $\mathsf{itr} \in \mathsf{EC}$:

Transfer of keys (encoding labels) and masked inputs: For each masked input b_w held by two garblers, do the following:
- Each P_g (owner of b_w) broadcasts $o[b]_w \oplus \mathsf{ran}_{g4}$. P_4 unmasks ran_{g4} and obtains b_w from $o[b_w]$ (if valid). Else, P_4 broadcasts $o[\mathsf{ran}]_{g4}$. Each party uses $o[\mathsf{ran}]_{g4}$ (if valid) to verify the validity of $o[b]_w$ and determines the corrupt party (say P_g). Run $\mathsf{3PC}$ with $\mathbb{C}^3 = \mathbf{P} \setminus \{P_g\}$.
- P_g sends $o[\mathsf{k}]^h_{w,b_w}, h \in \mathcal{S}_g$ to P_4 to allow P_4 to compute k^h_{w,b_w}. If $o[\mathsf{k}]^h_{w,b_w}$ sent by P_g is invalid, P_4 broadcasts $o[\mathsf{ran}]_{g4}$ and (Con, P_4, P_g). Garbler $P_j, j \neq g$ computes b_w (using $o[b]_w$) and broadcasts $o[\mathsf{k}]^h_{w,b_w}, h \in \mathcal{S}_j$. If an opening is still invalid (which can be checked by all), honest parties conclude the presence of two corrupt parties and abort. Else, P_4 uses valid openings to obtain $k^j_{w,b_w}, j \in [3]$.

Evaluation:
- Let $\mathsf{GC} = \mathsf{GC}^1||\mathsf{GC}^2||\mathsf{GC}^3$ and $\mathbf{X}$ be the encoding labels. P_4 evaluates GC to obtain $\mathbf{Y} = \{k^g_w\}_{g\in[3]}$ and $(y \oplus \lambda_w)$ for output wire w. P_4 broadcasts $\mathbf{Y}$. If not, run $\mathsf{3PC}$ with $\mathbb{C}^3 = \mathbf{P} \setminus \{P_4\}$.
- Each $P_i \in \mathbf{P}$ computes $\lambda_w = \oplus_{g\in[3]}\lambda^g_w$ using mask shares sent in *input-insensitive* phase and output y by unmasking λ_w.
- Let output obtained in iteration itr be denoted by y_{itr}. Output the majority value among $\{y_{\mathsf{itr}}\}_{\mathsf{itr}\in\mathsf{EC}}$.

Figure 11.5: Protocol $\pi_{\mathsf{bobw.god}}$

11.2 Security Proof

In this section, we provide the formal theorem and the elaborate the corresponding security proof.

Theorem 5. *The protocol $\pi_{\mathsf{bobw.god}}$ (Fig 11.5) securely realizes the functionalities $\mathcal{F}_{\mathsf{god}}$ (Fig 2.1) and $\mathcal{F}_{\mathsf{uAbort}}$ (Fig 2.3) in the standard model against one active corruption and two active corruptions respectively, assuming enhanced trapdoor permutation.*

Proof. The proof is presented by giving simulators separately for the honest majority and the dishonest majority case. Let $\mathcal{C}$ be the set of corrupt parties and $\mathcal{H} = [4] \setminus \mathcal{C}$ be the set of honest parties. The honest majority simulator (with 1 corruption) is denoted by $\mathcal{S}^{a}_{\mathsf{hm.bobw}}$ where P_a is the corrupt party while the dishonest majority simulator (with 2 corruptions) is denoted by $\mathcal{S}^{ab}_{\mathsf{dm.bobw}}$ where P_a and P_b are the two corrupt parties.

11.2.1 Honest Majority

Let $\mathcal{A}$ be a malicious adversary corrupting 1 party in an execution of $\pi_{\mathsf{bobw.god}}$. We discuss the honest majority simulator for two cases: (a) $\mathcal{S}^{1}_{\mathsf{hm.bobw}}$ when a garbler (say P_1) is corrupt and remaining parties are honest, (b) $\mathcal{S}^{4}_{\mathsf{hm.bobw}}$ when evaluator P_4 is corrupt and all garblers are honest. We now describe the simulator running an ideal-world of the GOD functionality $\mathcal{F}_{\mathsf{god}}$ (Fig 2.1) whose behaviour simulates the behaviour of $\mathcal{A}$.

Corrupt P_1: $\mathcal{C} = \{P_1\}$ and $\mathcal{H} = \{P_2, P_3, P_4\}$. $\mathcal{S}^{1}_{\mathsf{hm.bobw}}$ playing the role of parties in $\mathcal{H}$ works as follows:

For corrupt input x_1:

- Receive broadcast values $(\mathsf{pp}_1, c[\mathsf{in}]_{1i})$ for $i \in \mathsf{ind}(\mathbf{P}_1)$ and $o[\mathsf{in}]_{1i} \oplus \mathsf{ran}_{1i}$. Unmask the latter on behalf of P_i using ran_{1i} to obtain $o[\mathsf{in}]_{1i}$ and compute $x_{1i} \leftarrow \mathsf{Open}(\mathsf{pp}_1, c[\mathsf{in}]_{1i}, o[\mathsf{in}]_{1i})$.
- If, for some $i \in \mathsf{ind}(\mathbf{P}_1)$, $o[\mathsf{in}]_{1i}$ is invalid, broadcast $o[\mathsf{ran}]_{1i}$ on behalf of P_i. Run $\mathsf{3PC}()$ locally on behalf of honest P_2, P_3, P_4 and give output to P_1^*.

For honest input x_2:

- On behalf of P_2: sample random x_{21} and compute $(c[\mathsf{in}]_{21}, o[\mathsf{in}]_{21}) \leftarrow \mathsf{Com}(\mathsf{pp}_2, x_{21})$. Broadcast $(\mathsf{pp}_2, c[\mathsf{in}]_{21})$ and $o[\mathsf{in}]_{21} \oplus \mathsf{ran}_{12}$.
- If P_1 broadcasts $o[\mathsf{ran}]_{12}$, run $\mathsf{3PC}()$ locally on behalf of honest P_2, P_3, P_4 and give output to P_1^*.
- Also, sample dummy x_{2i} for $i \in \{3, 4\}$ and compute $(c[\mathsf{in}]_{2i}, o[\mathsf{in}]_{2i}) \leftarrow \mathsf{Com}(\mathsf{pp}_2, x_{2i})$. Broadcast $(\mathsf{pp}_2, c[\mathsf{in}]_{2i})$ and $o[\mathsf{in}]_{2i} \oplus \mathsf{ran}_{2i}$.

Figure 11.6: Simulator $\mathcal{S}^{1}_{\mathsf{ICom.hm}}$

- Act honestly on behalf of P_3 for the commitment instance between P_1 as sender and P_3 as receiver to obtain seed s_2.
- Sample random s_3 and act honestly on behalf of P_2 for the commitment instance between P_2 as sender and P_1 as receiver.

- Sample random s_1 and act honestly on behalf of P_3 and P_2 for the commitment instance between P_3 as sender and P_2 as receiver.

Figure 11.7: Simulator $\mathcal{S}^1_{\mathsf{seed.hm}}$

- Receive broadcast values $(\mathsf{pp}_k, c[\mathsf{ran}]_{ki})$ for $i > k$. Receive $o[\mathsf{ran}]_{ki}$ on behalf of honest P_i. Abort if opening is invalid. Else, compute $\mathsf{ran}_{ki} \leftarrow \mathsf{Open}(\mathsf{pp}_k, c[\mathsf{ran}]_{ki}, o[\mathsf{ran}]_{ki})$.
- On behalf of honest $P_i, i \in \mathsf{ind}(\mathbf{P}_k)$ for $i < k$, sample ran_{ik}, compute $(c[\mathsf{ran}]_{ik}, o[\mathsf{ran}]_{ik}) \leftarrow \mathsf{Com}(\mathsf{pp}_i, \mathsf{ran}_{ik})$. Broadcast $(\mathsf{pp}_k, c[\mathsf{ran}]_{ik})$.

Figure 11.8: Simulator $\mathcal{S}^k_{\mathsf{mask.hm}}$

- On behalf of $P_j, j \in [4] \setminus \{i\}$: Sample $\rho_j \leftarrow \{0,1\}^s$ and act as honest committer in ExtCom with P_{j+1} as receiver.
- On behalf of P_{i+1}: Act as honest receiver for the commitment instance between P_i as sender and P_{i+1} as receiver.

Figure 11.9: Extraction of challenge-string $\mathcal{S}^i_{\mathsf{CCstring.hm}}$

Seed and mask distribution: Run $\mathcal{S}^1_{\mathsf{seed.hm}}$ (Fig 11.7) and $\mathcal{S}^1_{\mathsf{mask.hm}}$ (Fig 11.8).

Input-sharing: Run $\mathcal{S}^1_{\mathsf{ICom.hm}}$ (Fig 11.6). Compute $x_1 = x_{12} \oplus x_{13} \oplus x_{14}$. Invoke $\mathcal{F}_{\mathsf{god}}$ (Fig 2.1) with (input, x_1) on behalf of P_1^* to obtain y.

Cut-and-choose Challenge String: Run $\mathcal{S}^1_{\mathsf{CCstring.hm}}$ (Fig 11.9). Run the following three phases for iterator $\mathsf{itr} \in [s]$:

(I) Input-insensitive Phase: Run this phase for each $\mathsf{itr} \in [s]$:

- On behalf of honest $P_g, g \in \{2,3\}$ and $h \in \mathcal{S}_g$, do the following steps honestly: Broadcast λ^h_w for output wire w. For every *input* wire w w.r.t. x_w held by two garblers: for each P_j holding x_w, $P_g, g \neq j$ broadcasts λ^j_w. (If P_4 holds x_w, then P_g broadcasts $\lambda^l_w, l \in \mathcal{S}_g$). For every input wire w, let $k^h_{w,0}$ and $k^h_{w,1}$ denote the two keys derived from seed s_h. For $b \in \{0,1\}$, compute commitments as: $(c[\mathsf{k}]^h_{w,b}, o[\mathsf{k}]^h_{w,b}) \leftarrow \mathsf{Com}(\mathsf{pp}^h, k^h_{w,b})$ and broadcasts $\{c[\mathsf{k}]^h_{w,b}\}$.
- If P_1 broadcasts different copies of λ^h_w (for output wire w) or λ^h_w or $c[\mathsf{k}]^h_{w,b}$ (for some *input* wire w and $b \in \{0,1\}$) for $h \in \mathcal{S}_1$ than what was computed by honest $P_g, g \in \mathcal{S}_h$, then on behalf of P_g: broadcast $o[\mathsf{s}]_h$. Run $\mathsf{3PC}$ locally on behalf of honest P_2, P_3, P_4 and give output to P_1^*.
- For input wire w owned by honest $P_g, g \in \{2,3,4\}$: compute $\lambda_w = (\oplus_{h \in [3]})\lambda^h_w$ using the knowledge of all seeds.

Commitment on masked inputs: On behalf of $P_i, i \in \{2,3,4\}$: For input wire w, with masked input b_w held by parties P_i, P_j, compute $(c[b]_w, o[b]_w) = \mathsf{Com}(\mathsf{pp}^{ij}, b_w)$ (using randomness derived

from shared ran_{ij}). Broadcast $(\mathsf{pp}^{ij}, c[b]_w)$. For b_w (w.r.t. x_{1i}) held by P_1, if P_1 broadcasts different values than computed by P_i, broadcast $o[\mathsf{ran}]_{1i}$, $\{o[\mathsf{s}]_l\}_{l\in[3]}$ and $o[\mathsf{in}]_{1i}$. Run $\mathsf{3PC}$ locally on behalf of honest P_2, P_3, P_4 and give output to P_1^*.

Oblivious Transfer: For wire w corresponding to shares owned by P_1 and P_4, say x_{14}:

- Receive $o[k]^h_{w,b_w}$ where $h \in \{2,3\}$ on behalf of P_4 from P_1 and compute k^h_{w,b_w}.
- If the opening is invalid: Broadcast $o[b]_w$ and (Con, P_4, P_1) on behalf of P_4. Broadcast $o[\mathsf{k}]^h_{w,b_w}$ on behalf of $P_g, g \in \{2,3\}$ for $h \in \mathcal{S}_g$.
- Else, $\mathcal{F}_{\mathsf{OT}}$ is called by P_1 (as receiver) and P_2 (as sender). Receive $o[k]^1_{w,b_w}$ from P_1 on behalf of P_4.

Transfer of GC:

- Behave honestly on behalf of $P_g, g \in \{2,3\}$ to realize $\mathcal{F}_{\mathsf{GCMod}}$ using seeds chosen in seed distribution phase. If any instance of $\pi_{\mathsf{aot.bobw}}$ returns $\mathbb{C}^3$, run $\mathsf{3PC}$ locally on behalf of honest P_2, P_3, P_4 and give output to P_1^*.
- Broadcast $\{\mathsf{GC}^h\}$ on behalf of $P_g, g \in \{2,3\}$ and $h \in \mathcal{S}_g$. If P_1 broadcasts different copy of GC^h for $h \in \mathcal{S}_1$ than what was computed by honest $P_g, g \in \mathcal{S}_h$, then on behalf of P_g: broadcast $o[\mathsf{s}]_h$. Run $\mathsf{3PC}$ locally on behalf of honest P_2, P_3, P_4 and give output to P_1^*.

(II) Cut-and-choose phase:

- Let the values broadcasted corresponding to seed s_g in the above three phases in iteration itr be denoted by $\mathbf{B}[g]_{\mathsf{itr}}$.
- On behalf of $P_j, j \in \{2,3,4\}$: receive opening broadcasted by P_1 and compute ρ_1. Also, run $\mathsf{ExtOpen}$ routine to broadcast opening for ρ_j. Compute $\rho = \oplus_{i\in[4]}\rho_i$.
- Let $\mathsf{CC} = \{k : \rho^k = 1\}$ where ρ^k is used to denote the k^{th} bit of ρ and $\mathsf{EC} = [s] \setminus \mathsf{CC}$.
- For $\mathsf{itr} \in \mathsf{CC}$, the following is done:
  - On behalf of $P_g, g \in \{2,3\}$: broadcast the openings of seed $\mathsf{s}_{(g \mod 3)+1}$ and masks ran_{gh} for $(h > g)$. Receive openings of seed s_2 and masks ran_{1j} for $j \in \{2,3,4\}$.
  - If the opening broadcasted by P_1 is incorrect, run $\mathsf{3PC}$ locally on behalf of honest P_2, P_3, P_4 and give output to P_1^*.

Input-sensitive phase: Run the following three phases for $\mathsf{itr} \in \mathsf{EC}$:

Transfer of keys (encoding labels) and masked inputs: For wire w corresponding to shares possessed by two garblers:

- On behalf of $P_g, g \in \{2,3\}$: If P_g is an owner of w, broadcast $o[b]_w \oplus \mathsf{ran}_{g4}$.
- If P_1 is an owner of w, receive $o[\mathsf{k}]^h_{w,b_w}$ where $h \in \mathcal{S}_1$ from P_1 on behalf of P_4.
- If opening sent by P_1 is invalid, broadcast $o[\mathsf{ran}]_{14}$ and (Con, P_1, P_4). Broadcast $o[\mathsf{k}]^h_{w,b_w}$ where $h \in \mathcal{S}_g$ on behalf of $P_g, g \in \{2,3\}$.

Evaluation:

- Using the knowledge of all seeds $\mathsf{s}_g, g \in [3]$: compute $z = y \oplus \lambda_w$ and $\mathbf{Y} = \{k^g_{w,z}\}_{g\in[3]}$ for output wire w. Broadcast $\mathbf{Y}$ on behalf of P_4.

Figure 11.10: Simulator $\mathcal{S}^1_{\mathsf{hm.bobw}}$

Security against active P_1^*: We now argue that $\text{IDEAL}_{\mathcal{F}_{\mathsf{god}},\mathcal{S}^1_{\mathsf{hm.bobw}}} \overset{c}{\approx} \text{REAL}_{\pi_{\mathsf{bobw.god}},\mathcal{A}}$ when an adversary $\mathcal{A}$ corrupts a single party P_1 (honest majority). The views are shown to be indistinguishable via a series of intermediate hybrids.

– HYB_0: Same as $\text{REAL}_{\pi_{\mathsf{bobw.god}},\mathcal{A}}$.

– HYB_1: Same as HYB_0 except: For share x_{jk} for $j \in [4] \setminus \{1\}, k \neq 1$ (i.e. the share that the adversary doesn't get access to), replace $c[\mathsf{in}]_{jk}$ with the commitment of a dummy value.

– HYB_2: Same as HYB_1 except: During the OT phase, invoke $\mathcal{F}_{\mathsf{OT}}$ for OTs where P_1 acts as receiver to obtain $o[k]^1_{w,b_w}$ for wire w corresponding to shares possessed by P_1 and P_4.

– HYB_3: Same as HYB_2 except: Compute $z = y \oplus \lambda_w$ and $\mathbf{Y} = \{k^g_{w,z}\}_{g\in[3]}$ instead of running the Evaluation Phase of garbling. Here, y is the output after invoking $\mathcal{F}_{\mathsf{god}}$ with (input, x_1).

– HYB_4: Same as HYB_3 except: in case of a P_1 identified to be corrupt, compute y locally between P_2, P_3, P_4 (from all the known inputs) instead of running $\mathsf{3PC}$.

Note that $\text{HYB}_4 = \text{IDEAL}_{\mathcal{F}_{\mathsf{god}},\mathcal{S}^1_{\mathsf{hm.bobw}}}$. Next, we show that each pair of hybrids is computationally indistinguishable as follows:

$\text{HYB}_0 \overset{c}{\approx} \text{HYB}_1$: The only difference between the hybrids is that in HYB_2, the commitment for shares x_{jk} for $j \in [4]\setminus\{1\}, k \neq 1$ are replaced by commitments of dummy values. Note that these are the shares whose openings are not revealed to the adversary. Hence, the indistinguishability follows from the hiding property of the commitment scheme.

$\text{HYB}_1 \overset{c}{\approx} \text{HYB}_2$: Indistinguishability of hybrids follows from the security of the underlying OT scheme.

$\text{HYB}_2 \overset{c}{\approx} \text{HYB}_3$: The indistinguishability follows from the correctness of the garbling scheme since $\mathbf{Y}$ computed using the Evaluation Phase of garbling would also result in $\mathbf{Y} = \{k^g_{w,y\oplus\lambda_w}\}_{g\in[3]}$ where $y = f(x_1, x_2, x_3, x_4)$.

$\text{HYB}_3 \overset{c}{\approx} \text{HYB}_4$: The indistinguishability follows from the correctness of $\mathsf{3PC}$.

Corrupt P_4: $\mathcal{C} = \{P_4\}$ and $\mathcal{H} = \{P_1, P_2, P_3\}$. $\mathcal{S}^4_{\mathsf{hm.bobw}}$ playing the role of parties in $\mathcal{H}$ works as follows:

Act honest on behalf of both the sender and the receiver for the three commitment instances run as: a) P_1 as sender and P_3 as receiver for seed s_2, b) P_2 as sender and P_1 as receiver for seed s_3, c) P_3 as sender and P_2 as receiver for seed s_1.

Figure 11.11: Simulator $\mathcal{S}^4_{\mathsf{seed.hm}}$

- On behalf of $P_j, j \in [3]$: Sample $\rho_j \leftarrow \{0,1\}^s$ and act as honest committer in ExtCom with P_{j+1} as receiver.
- For the commitment instance between P_4 as sender and P_1 as receiver to commit to string ρ_4:
  - Run the ExtCom protocol with P_4 as sender and honest P_1 as receiver, run rounds 1-3 and broadcast their messages $(\mathsf{extcom}^1_1, \mathsf{extcom}^1_2, \mathsf{extcom}^1_3)$.
  - Rewind the adversary to the end of round 1 for P_4 and P_1 to rerun rounds 2-3 and broadcast $(\mathsf{extcom}^2_2, \mathsf{extcom}^2_3)$.
  - Run extractor algorithm $\mathsf{Extract}$ of the commitment scheme as in Fig 9.2 using inputs $(\mathsf{extcom}^1_1, \{\mathsf{extcom}^i_2, \mathsf{extcom}^i_3\}_{i\in[2]})$ to extract the committed string ρ_4.

Figure 11.12: Extraction of challenge-string $\mathcal{S}^4_{\mathsf{CCstring.hm}}$

Seed and mask distribution: Run $\mathcal{S}^4_{\mathsf{seed.hm}}$ (Fig 11.11) and $\mathcal{S}^4_{\mathsf{mask.hm}}$ (Fig 11.8).

Input-sharing: Run $\mathcal{S}^4_{\mathsf{ICom.hm}}$ (Fig 11.6). Compute $x_4 = x_{41} \oplus x_{42} \oplus x_{43}$. Invoke $\mathcal{F}_{\mathsf{god}}$ (Fig 2.1) with (input, x_4) on behalf of P_4^* to obtain y.

Cut-and-choose Challenge String: Run $\mathcal{S}^4_{\mathsf{CCstring.hm}}$ (Fig 11.12) to compute $\rho = \oplus_{i\in[4]}\rho_i$. Let $\mathsf{CC} = \{k : \rho^k = 1\}$ where ρ^k is used to denote the k^{th} bit of ρ and $\mathsf{EC} = [s] \setminus \mathsf{CC}$.
Run the following three phases for iterator $\mathsf{itr} \in [s]$:

(I) Input-insensitive phase:
- On behalf of honest $P_g, g \in [3]$ and $h \in \mathcal{S}_g$, do the following steps honestly: Broadcast λ^h_w for output wire w. For every *input* wire w w.r.t. x_w held by two garblers: for each P_j holding x_w, $P_g, g \neq j$ broadcasts λ^j_w. (If P_4 holds w, P_g broadcasts $\lambda^l_w, l \in \mathcal{S}_g$). For every input wire w, let $k^h_{w,0}$ and $k^h_{w,1}$ denote the two keys derived from seed s_h. For $b \in \{0,1\}$, compute commitments as: $(c[\mathsf{k}]^h_{w,b}, o[\mathsf{k}]^h_{w,b}) \leftarrow \mathsf{Com}(\mathsf{pp}^h, k^h_{w,b})$ and broadcasts $\{c[\mathsf{k}]^h_{w,b}\}$.
- For input wire w owned by honest $P_g, g \in \{2,3,4\}$: compute $\lambda_w = (\oplus_{h\in[3]})\lambda^h_w$ using the knowledge of all seeds.

Commitment on masked inputs: On behalf of $P_i, i \in [3]$: For input wire w, with masked input b_w held by parties P_i, P_j, compute $(c[b]_w, o[b]_w) = \mathsf{Com}(\mathsf{pp}^{ij}, b_w)$ (using randomness derived from shared ran_{ij}). Broadcast $(\mathsf{pp}^{ij}, c[b]_w)$. For b_w (say w.r.t. x_{i4}) held by P_4, if P_4 broadcasts different

values than computed by P_i, broadcast $o[\mathsf{ran}]_{i4}$, $\{o[\mathsf{s}]_l\}_{l\in[3]}$ and $o[\mathsf{in}]_{i4}$. Run $\mathsf{3PC}$ locally on behalf of honest P_1, P_2, P_3 and give output to P_4^*.

Oblivious Transfer: For wire w corresponding to shares owned by $P_g, g \in [3]$ and P_4, say x_{14}:
- Send $o[k]^h_{w,b_w}$ where $h \in \mathcal{S}_g$ to P_4 on behalf of P_g.
- $\mathcal{F}_{\mathsf{OT}}$ is called by P_ℓ ($\ell = 2$ if $g = 3; \ell = 3$ otherwise) (as sender) and P_4 (as receiver).
- Send $o[k]^g_{w,b_w}$ to P_4 on behalf of P_g.

Transfer of GC:
- For $\mathsf{itr} \in \mathsf{CC}$, i.e. for GC that will be opened according to the challenge string, behave honestly on behalf of $P_g, g \in [3]$ to realise $\mathcal{F}_{\mathsf{GCMod}}$ using seeds chosen in seed distribution phase.
- For $\mathsf{itr} \in \mathsf{EC}$, i.e. for GCs that will be evaluated, compute $z = y \oplus \lambda_w$ for the output wire w. Construct a simulated $\mathsf{GC}^1||\mathsf{GC}^2||\mathsf{GC}^3$ using the knowledge of all seeds such that each ciphertext for the output gate of GC^g encrypts the same output key $k^g_{w,z}$.
- Broadcast $\{\mathsf{GC}^h\}$ on behalf of $P_g, g \in [3]$ and $h \in \mathcal{S}_g$.

(II) Cut-and-choose phase:
- Let the values broadcasted corresponding to seed s_g in the above three phases in iteration itr be denoted by $\mathbf{B}[g]_{\mathsf{itr}}$.
- On behalf of $P_j, j \in [3]$: receive opening broadcasted by P_4. If opening sent by P_4 is invalid, invoke simulator for $\mathsf{3PC}$ (P_4 corrupt). Also, run $\mathsf{ExtOpen}$ routine to broadcast opening for ρ_j.
- For $\mathsf{itr} \in \mathsf{CC}$, the following is done:
  - On behalf of P_1, broadcast openings of seed s_2 and masks ran_{12}, ran_{13} and ran_{14}.
  - On behalf of P_2, broadcast openings of seed s_3 and masks ran_{23} and ran_{24}.
  - On behalf of P_3, broadcast openings of seed s_1 and masks ran_{34}.

Input-sensitive phase: Run the following three phases for $\mathsf{itr} \in \mathsf{EC}$:

Transfer of keys (encoding labels) and masked inputs: For wire w corresponding to shares possessed by two garblers, do the following on behalf of $P_g, g \in [3]$:
- If P_g is an owner of w, broadcast $o[b]_w \oplus o[\mathsf{ran}]_{g4}$.
- If P_g is an owner of w, send $o[\mathsf{k}]^h_{w,b_w}$ where $h \in \mathcal{S}_g$ to P_4 on behalf of P_g.

Evaluation:
- Receive $\mathbf{Y}$ on behalf of $P_g, g \in [3]$ as broadcasted by P_4. If P_4 does not broadcast anything or if $\mathbf{Y} \neq \{k^h_{w,z}\}_{h\in[3]}$, Run $\mathsf{3PC}$ locally on behalf of honest P_1, P_2, P_3 and give output to P_4^*.

Figure 11.13: Simulator $\mathcal{S}^4_{\mathsf{hm.bobw}}$

Security against active P_4^*: We now argue that $\text{IDEAL}_{\mathcal{F}_{\mathsf{god}},\mathcal{S}^4_{\mathsf{hm.bobw}}} \overset{c}{\approx} \text{REAL}_{\pi_{\mathsf{bobw.god}},\mathcal{A}}$ when an adversary $\mathcal{A}$ corrupts a single party P_4 (honest majority). The views are shown to be indistin-

guishable via a series of intermediate hybrids.

– HYB_0: Same as $\text{REAL}_{\pi_{\mathsf{bobw.god}},\mathcal{A}}$.

– HYB_1: Same as HYB_0 except: Rerun rounds 2-3 of extractable commitment for the commitment of the challenge string for cut-and-choose (with P_4 as sender and P_1 as receiver) to extract ρ_4. Run the subsequent rounds same as HYB_0.

– HYB_2: Same as HYB_1 except: For share x_{jk} for $j \in [3], k \neq 4$ (i.e. the share that the adversary doesn't get access to), replace $c[\mathsf{in}]_{jk}$ with the commitment of a dummy value.

– HYB_3: Same as HYB_2 except: For $\mathsf{itr} \in \mathsf{EC}$ i.e. for GCs that will be evaluated, construct simulated GC using knowledge of all seeds (instead of constructing an honest GC), in such a way that each ciphertext for the output gate encrypts the same output key which corresponds to $b_w = y \oplus \lambda_w$ where y is obtained after having invoked $\mathcal{F}_{\mathsf{god}}$ and λ_w is known from the information of all seeds.

– HYB_4: Same as HYB_3 except: In HYB_4, $\mathbf{Y}$ is deemed to be invalid if there does not exist a bit b_w such that for each $j \in \mathcal{S}_g$, k_w^j obtained from $\mathbf{Y}$ matches k_{w,b_w}^j while in HYB_5, it $\mathbf{Y}$ is deemed invalid if it is not the one that was encrypted in the simulated GC.

– HYB_5: Same as HYB_4 except: in case of a P_1 identified to be corrupt, compute y locally between P_2, P_3, P_4 (from all the known inputs) instead of running $\mathsf{3PC}$.

Note that $\text{HYB}_4 = \text{IDEAL}_{\mathcal{F}_{\mathsf{god}},\mathcal{S}^4_{\mathsf{hm.bobw}}}$. Next, we show that each pair of hybrids is computationally indistinguishable as follows:

$\text{HYB}_0 \overset{c}{\approx} \text{HYB}_1$: The only difference is that in HYB_1, rounds 2-3 of extractable commitment for the commitment of challenge string for cut-and-choose are rerun to extract ρ_4. Note that the view of the adversary doesn't change across rewinds.

$\text{HYB}_1 \overset{c}{\approx} \text{HYB}_2$: The only difference between the hybrids is that in HYB_2, the commitment for shares x_{jk} for $j \in [4]\backslash\{1\}, k \neq 1$ are replaced by commitments of dummy values. Note that these are the shares whose openings are not revealed to the adversary. Hence, the indistinguishability follows from the hiding property of the commitment scheme.

$\text{HYB}_2 \overset{c}{\approx} \text{HYB}_3$: Indistinguishability of hybrids follows from reduction to the security of the underlying garbling scheme which breaks down to the security of PRF. Also, it is ensured that the GCs that are going to be opened for verification have been constructed correctly, so that the adversary cannot distinguish w.r.t. those GCs.

$\text{HYB}_3 \overset{c}{\approx} \text{HYB}_4$: Indistinguishability follows for the two different notions of validity of $\mathbf{Y}$ because a $\mathbf{Y}$ valid according to condition in HYB_5 is valid according to condition in HYB_4. Also

a $\mathbf{Y}$ invalid according to condition in HYB_5 can possibly be valid according to condition in HYB_4 only if P_4 could forge the other output keys which is not possible with non-negligible probability according to the authenticity of the garbling scheme.

$\text{HYB}_3 \overset{c}{\approx} \text{HYB}_4$: The indistinguishability follows from the correctness of $\mathsf{3PC}$.

11.2.2 Dishonest Majority

Let $\mathcal{A}$ be a malicious adversary corrupting 2 parties in an execution of $\pi_{\mathsf{bobw.god}}$. We discuss the dishonest majority simulator for two cases: (a) $\mathcal{S}^{12}_{\mathsf{dm.bobw}}$ when two garblers (say P_1, P_2) are corrupt and remaining parties P_3, P_4 are honest, (b) $\mathcal{S}^{14}_{\mathsf{dm.bobw}}$ when a garbler (say P_1) and evaluator P_4 are corrupt and garblers P_2, P_3 are honest. We now describe the simulator running an ideal-world of the GOD functionality $\mathcal{F}_{\mathsf{god}}$ (Fig 2.1) whose behaviour simulates the behaviour of $\mathcal{A}$.

Corrupt P_1, P_2: $\mathcal{C} = \{P_1, P_2\}$ and $\mathcal{H} = \{P_3, P_4\}$. $\mathcal{S}^{12}_{\mathsf{dm.bobw}}$ playing the role of parties in $\mathcal{H}$ works as follows:

For corrupt input x_1:

- Receive broadcast values $(\mathsf{pp}_1, c[\mathsf{in}]_{1i})$ for $i \in \mathsf{ind}(\mathbf{P}_1)$ and $o[\mathsf{in}]_{1i} \oplus \mathsf{ran}_{1i}$. Unmask the latter on behalf of honest P_j for $j \in \{3,4\}$ using ran_{1j} to obtain $o[\mathsf{in}]_{1j}$ and compute $x_{1j} \leftarrow \mathsf{Open}(\mathsf{pp}_1, c[\mathsf{in}]_{1j}, o[\mathsf{in}]_{1j})$.
- If, for some j, $o[\mathsf{in}]_{1j}$ is invalid, broadcast $o[\mathsf{ran}]_{1j}$ on behalf of P_j for $j \in \{3,4\}$. Invoke simulator for $\mathsf{3PC}()$ with P_2, P_3, P_4 as participating parties (with P_2 as the corrupt party).

For honest input x_3:

- On behalf of P_3: sample random x_{3i} for $i \in [2]$ and compute $(c[\mathsf{in}]_{3i}, o[\mathsf{in}]_{3i}) \leftarrow \mathsf{Com}(\mathsf{pp}_3, x_{3i})$. Broadcast $(\mathsf{pp}_3, c[\mathsf{in}]_{3i})$ and $o[\mathsf{in}]_{3i} \oplus \mathsf{ran}_{i3}$.
- If P_i broadcasts $o[\mathsf{ran}]_{i3}$, invoke simulator for $\mathsf{3PC}()$ with P_j, P_3, P_4 where $j = [2] \setminus \{i\}$ as participating parties (with P_j as the corrupt party).
- Also, sample dummy x_{34} and compute $(c[\mathsf{in}]_{34}, o[\mathsf{in}]_{34}) \leftarrow \mathsf{Com}(\mathsf{pp}_3, x_{34})$. Broadcast $(\mathsf{pp}_3, c[\mathsf{in}]_{34})$ and $o[\mathsf{in}]_{34} \oplus \mathsf{ran}_{34}$.

Figure 11.14: Simulator $\mathcal{S}^{12}_{\mathsf{ICom.dm}}$

- Act honestly on behalf of P_3 for the commitment instance between P_1 as sender and P_3 as receiver to obtain seed s_2.
- Sample random s_1 and act honestly on behalf of P_3 for the commitment instance between P_3 as sender and P_2 as receiver.
- For the commitment instance between P_1 as sender and P_2 as receiver to commit to seed s_3:
  - Run the ExtCom protocol where P_1 and P_2 run rounds 1-3 and broadcast their messages

$(\mathsf{extcom}_1^1, \mathsf{extcom}_2^1, \mathsf{extcom}_3^1)$.

- Rewind the adversary to the end of round 1 for P_1 and P_2 to rerun rounds 2-3 and broadcast $(\mathsf{extcom}_2^2, \mathsf{extcom}_3^2)$.
- On behalf of P_3, Run extractor algorithm $\mathsf{Extract}$ of the commitment scheme as in Fig 9.2 using inputs $(\mathsf{extcom}_1^1, \{\mathsf{extcom}_2^i, \mathsf{extcom}_3^i\}_{i\in[2]})$ to extract the committed seed s_3.

Figure 11.15: Simulator $\mathcal{S}^{12}_{\mathsf{seed.dm}}$

- Receive broadcast values $(\mathsf{pp}_k, c[\mathsf{ran}]_{ki})$ for $i > k$. Receive $o[\mathsf{ran}]_{ki}$ on behalf of honest P_i. Abort if opening is invalid. Else, compute $\mathsf{ran}_{ki} \leftarrow \mathsf{Open}(\mathsf{pp}_k, c[\mathsf{ran}]_{ki}, o[\mathsf{ran}]_{ki})$.
- On behalf of honest $P_i \in \mathbf{P} \setminus \{P_k, P_\ell\}$ for $i < k$, sample ran_{ik}, compute $(c[\mathsf{ran}]_{ik}, o[\mathsf{ran}]_{ik}) \leftarrow \mathsf{Com}(\mathsf{pp}_i, \mathsf{ran}_{ik})$. Broadcast $(\mathsf{pp}_k, c[\mathsf{ran}]_{ik})$.
- Receive broadcast values $(\mathsf{pp}_\ell, c[\mathsf{ran}]_{\ell i})$ for $i > \ell$. Receive $o[\mathsf{ran}]_{\ell i}$ on behalf of honest P_i. Abort if opening is invalid. Else, compute $\mathsf{ran}_{\ell i} \leftarrow \mathsf{Open}(\mathsf{pp}_\ell, c[\mathsf{ran}]_{\ell i}, o[\mathsf{ran}]_{\ell i})$.
- On behalf of honest $P_i \in \mathbf{P} \setminus \{P_k, P_\ell\}$ for $i < \ell$, sample $\mathsf{ran}_{i\ell}$, compute $(c[\mathsf{ran}]_{i\ell}, o[\mathsf{ran}]_{i\ell}) \leftarrow \mathsf{Com}(\mathsf{pp}_i, \mathsf{ran}_{i\ell})$. Broadcast $(\mathsf{pp}_\ell, c[\mathsf{ran}]_{i\ell})$.

Figure 11.16: Simulator $\mathcal{S}^{k\ell}_{\mathsf{mask.dm}}$

- On behalf of $P_j, j \in \{3, 4\}$: Sample $\rho_j \leftarrow \{0,1\}^s$ and act as honest committer in ExtCom with P_{j+1} as receiver.
- On behalf of P_3: Act as honest receiver for the commitment instance between P_2 as sender and P_3 as receiver.

Figure 11.17: Simulator $\mathcal{S}^{12}_{\mathsf{CCstring.dm}}$

Seed and mask distribution: Run $\mathcal{S}^{12}_{\mathsf{seed.dm}}$ (Fig 11.15) and $\mathcal{S}^{12}_{\mathsf{mask.dm}}$ (Fig 11.16). Extract seed s_3.

Input-sharing: Run $\mathcal{S}^{12}_{\mathsf{ICom.dm}}$ (Fig 11.14).

Cut-and-choose Challenge String: Run $\mathcal{S}^{12}_{\mathsf{CCstring.dm}}$ (Fig 11.17).
Run the following three phases for iterator $\mathsf{itr} \in [s]$:

(I) Input-insensitive phase:

- On behalf of honest P_3 and $h \in \mathcal{S}_3$, do the following steps honestly: Broadcast λ_w^h for output wire w. For every *input* wire w, if w is owned by $P_g, g \in [2]$, broadcast λ_w^g. (If P_4 owns w, broadcast λ_w^h). For every input wire w, let $k_{w,0}^h$ and $k_{w,1}^h$ denote the two keys derived from seed s_h. For $b \in \{0,1\}$, compute commitments as: $(c[\mathsf{k}]_{w,b}^h, o[\mathsf{k}]_{w,b}^h) \leftarrow \mathsf{Com}(\mathsf{pp}^h, k_{w,b}^h)$ and broadcast $\{c[\mathsf{k}]_{w,b}^h\}$.
- If $P_i, i \in [2]$ broadcasts different copies of λ_w^h (for output wire w) or λ_w^h or $c[\mathsf{k}]_{w,b}^h$ (for some *input* wire w and $b \in \{0,1\}$) for $h \in \mathcal{S}_3$ than what was computed by honest P_3, then on behalf of P_3:

broadcast $o[\mathsf{s}]_h$. Invoke simulator for $\mathsf{3PC}()$ with P_j, P_3, P_4 where $P_j = [2] \setminus \{i\}$ as participating parties (with P_j as the corrupt party).

- If P_1 and P_2 broadcast different copies of λ_w^3 (for output wire w) or λ_w^3 or $c[\mathsf{k}]_{w,b}^3$ (for some *input* wire w and $b \in \{0,1\}$), then receive $o[\mathsf{s}]_3$ from P_1 and P_2 and conclude the party (out of the two) who misbehaved by locally computing the messages they broadcasted using $o[\mathsf{s}]_3$. Invoke simulator for $\mathsf{3PC}()$ accordingly.
- For input wire w owned by honest $P_g, g \in \{3,4\}$: compute $\lambda_w = (\oplus_{h\in[3]})\lambda_w^h$ using the knowledge of all seeds (including the extracted s_3).

Commitment on masked inputs:

- On behalf of $P_i, i \in \{3,4\}$: For input wire w, with masked input b_w held by parties $P_i, P_j, j \neq i$, compute $(c[b]_w, o[b]_w) = \mathsf{Com}(\mathsf{pp}^{ij}, b_w)$ (using randomness derived from shared ran_{ij}). Broadcast $(\mathsf{pp}^{ij}, c[b]_w)$.
- For input wire w (w.r.t. share x_{ij}) where $i \in \{3,4\}$ and $j \in [2]$: If P_j broadcasts different values than computed by P_i, broadcast $o[\mathsf{ran}]_{ji}$, $\{o[\mathsf{s}]_l\}_{l\in\mathcal{S}_i}$ and $o[\mathsf{in}]_{ij}$.
- For input wire w (w.r.t. share held by P_1, P_2, say x_{12}): if P_1, P_2 broadcast mismatching values, receive $o[\mathsf{ran}]_{12}, \{o[seed]_l\}_{l\in\mathcal{S}_j}$ broadcasted by $P_j, j \in [2]$ and conclude the party (out of the two) by locally computing the messages they broadcasted (say P_1 is corrupt). Invoke simulator for $\mathsf{3PC}()$ accordingly.

Oblivious Transfer: For wire w corresponding to shares owned by $P_g, g \in [3]$ and P_4, say x_{14}:

- Receive $o[k]_{w,b_w}^h$ where $h \in \mathcal{S}_1$ on behalf of P_4 from P_1 and compute k_{w,b_w}^h.
- If any opening sent by P_1 is invalid: Broadcast $o[b]_w$ and (Con, P_1, P_4) on behalf of P_4 and $o[\mathsf{k}]_{w,b_w}^h$ on behalf of P_3 for $h \in \mathcal{S}_3$. Receive $o[\mathsf{k}]_{w,b_w}^h$ broadcasted by P_2 for $h \in \mathcal{S}_2$. If any opening sent by P_2 is invalid, invoke $\mathcal{F}_{\mathsf{uAbort}}$ with $(\mathsf{input}, \perp)$ and set $y = \perp$.
- Else, receive $o[k]_{w,b_w}^1$ from P_1 on behalf of P_4 (after P_1 (as receiver) and P_2 (as sender) would have run an OT).

Transfer of GC:

- Behave honestly on behalf of P_3 to realise $\mathcal{F}_{\mathsf{GCMod}}$ using seeds $\mathsf{s}_1, \mathsf{s}_2$ chosen in seed distribution phase. If any instance of $\pi_{\mathsf{aot.bobw}}$ returns $\mathbb{C}^3$, invoke simulator for $\mathsf{3PC}()$ with $\mathbb{C}^3$ as participating parties (with $\mathbb{C}^3 \setminus \{P_3, P_4\}$ as the corrupt party).
- Broadcast $\{\mathsf{GC}^h\}$ on behalf of P_3 for $h \in \mathcal{S}_3$. If P_1 (or P_2) broadcasts different copies of GC^h than what was computed by honest P_3, then on behalf of P_3: broadcast $o[\mathsf{s}]_h$. Invoke simulator for $\mathsf{3PC}()$ accordingly.
- If P_1 and P_2 broadcast different copies of GC^3, then receive $o[\mathsf{s}]_3$ from P_1 and P_2 and conclude the party (out of the two) who misbehaved by locally computing GC^3. Invoke simulator for $\mathsf{3PC}()$ accordingly.

(II) Cut-and-choose phase:

- Let the values broadcasted corresponding to seed s_g in the above three phases in iteration itr be denoted by $\mathbf{B}[g]_{\mathsf{itr}}$.
- On behalf of $P_j, j \in \{3,4\}$: receive opening broadcasted by P_1, P_2 and compute ρ_1, ρ_2. Also, run $\mathsf{ExtOpen}$ routine to broadcast opening for ρ_j. Compute $\rho = \oplus_{i\in[4]}\rho_i$.
- Let $\mathsf{CC} = \{k : \rho^k = 1\}$ where ρ^k is used to denote the k^{th} bit of ρ and $\mathsf{EC} = [s] \setminus \mathsf{CC}$.
- For $\mathsf{itr} \in \mathsf{CC}$, the following is done:
  - On behalf of P_3, broadcast openings of seed s_1 and masks ran_{34}.
  - If any opening broadcasted by $P_g, g \in [2]$ is incorrect, run simulator for $\mathsf{3PC}$ with $\mathbb{C}^3 = \mathbf{P} \setminus \{P_g\}$.
  - Check whether $\mathbf{B}[3]_{\mathsf{itr}}$ sent by P_1, P_2 is consistent with s_3. If not, invoke $\mathcal{F}_{\mathsf{uAbort}}$ with $(\mathsf{input}, \perp)$ and set $y = \perp$.

(III) Input-sensitive phase: Run the following three phases for $\mathsf{itr} \in \mathsf{EC}$:

Transfer of keys (encoding labels) and masked inputs: For each masked input b_w held by two garblers, do the following:

- On behalf of P_3, if P_3 is an owner of w, broadcast $o[b]_w \oplus \mathsf{ran}_{34}$.
- For share x_{12} owned by P_1, P_2, receive $o[b]_w \oplus \mathsf{ran}_{i4}$ from $P_i, i \in [2]$. If both P_1, P_2 send invalid $o[b]_w$, on behalf of P_4: broadcast $o[\mathsf{ran}]_{14}$ and (Con, P_1, P_4). Invoke simulator for $\mathsf{3PC}()$ with P_2, P_3, P_4 as participating parties (with P_2 as the corrupt party).
- Else, use b_w received to compute $x_{12} = b_w \oplus (\oplus_{h\in[3]}\lambda_w^h)$ (from the knowledge of all seeds). Compute $x_1 = x_{12} \oplus x_{13} \oplus x_{14}$. Compute x_2 similarly.
- For share owned by $P_i, i \in [2]$, receive $o[\mathsf{k}]^h_{w,b_w}$ from P_i for $h \in \mathcal{S}_i$ on behalf of P_4.
- If opening sent by either party (say P_1) is invalid, broadcast $o[\mathsf{ran}]_{i4}$ and (Con, Pi, P_4). Broadcast $o[k]^h_{w,b_w}$ for $h \in \mathcal{S}_3$ on behalf of P_3. If $P_j, j \in [2] \setminus \{i\}$ broadcasts an invalid opening, invoke $\mathcal{F}_{\mathsf{uAbort}}$ with $(\mathsf{input}, \perp)$ and set $y = \perp$.

Evaluation:

- Invoke $\mathcal{F}_{\mathsf{uAbort}}$ (Fig 2.3) with $(\mathsf{input}, x_1), (\mathsf{input}, x_2)$ on behalf of P_1^*, P_2^* to obtain y.
- Using the knowledge of all seeds $\mathsf{s}_g, g \in [3]$ and y: compute $z = y \oplus \lambda_w$ and $\mathbf{Y} = \{k^g_{w,z}\}_{g\in[3]}$ for output wire w. Broadcast $\mathbf{Y}$ on behalf of P_4.

Figure 11.18: Simulator $\mathcal{S}^{12}_{\mathsf{dm.bobw}}$

Security against active P_1^* and P_2^*: We now argue that $\text{IDEAL}_{\mathcal{F}_{\mathsf{uAbort}},\mathcal{S}^{12}_{\mathsf{dm.bobw}}} \overset{c}{\approx} \text{REAL}_{\pi_{\mathsf{bobw.god}},\mathcal{A}}$ when an adversary $\mathcal{A}$ corrupts two parties P_1, P_2 (dishonest majority). The views are shown to be indistinguishable via a series of intermediate hybrids.

– HYB_0: Same as $\text{REAL}_{\pi_{\mathsf{bobw.god}},\mathcal{A}}$.

– HYB_1: Same as HYB_0 except: Rerun rounds 2-3 of extractable commitment (with P_2 as sender

and P_1 as receiver) in the seed-distribution phase to extract seed s_3. Run the subsequent rounds same as HYB_0.

– HYB_2: Same as HYB_1 except: For share x_{ij} for $i \in \{3,4\}, j = \{3,4\} \setminus \{i\}$ (i.e. the share that the adversary doesn't get access to), replace $c[\mathsf{in}]_{ij}$ with the commitment of a dummy value.

– HYB_3: Same as HYB_2 except: Compute $z = y \oplus \lambda_w$ and $\mathbf{Y} = \{k^g_{w,z}\}_{g\in[3]}$ instead of running the Evaluation Phase of garbling. Here, y is the output after invoking $\mathcal{F}_{\mathsf{god}}$ with (input, x_1) and λ_w is computed using the knowledge of all seeds.

– HYB_4: Same as HYB_3 except: in case of a $P_i, i \in [2]$ identified to be corrupt, invoke simulator for $\mathsf{3PC}$ instead of running the actual $\mathsf{3PC}$.

Note that $\text{HYB}_4 = \text{IDEAL}_{\mathcal{F}_{\mathsf{uAbort}}, \mathcal{S}^{12}_{\mathsf{dm.bobw}}}$. Next, we show that each pair of hybrids is computationally indistinguishable as follows:

$\text{HYB}_0 \overset{c}{\approx} \text{HYB}_1$: The only difference is that in HYB_1, rounds 2-3 of extractable commitment for the seed-distribution phase are rerun to extract s_3. Note that the view of the adversary doesn't change across rewinds.

$\text{HYB}_1 \overset{c}{\approx} \text{HYB}_2$: The only difference between the hybrids is that in HYB_2, the commitment for shares x_{ij} for $i \in \{3,4\}, j = \{3,4\} \setminus \{i\}$ are replaced by commitments of dummy values. Note that these are the shares whose openings are not revealed to the adversary. Hence, the indistinguishability follows from the hiding property of the commitment scheme.

$\text{HYB}_2 \overset{c}{\approx} \text{HYB}_3$: The indistinguishability follows from the correctness of the garbling scheme since $\mathbf{Y}$ computed using the Evaluation Phase of garbling would also result in $\mathbf{Y} = \{k^g_{w,y\oplus\lambda_w}\}_{g\in[3]}$ where $y = f(x_1, x_2, x_3, x_4)$.

$\text{HYB}_3 \overset{c}{\approx} \text{HYB}_4$: The indistinguishability follows from the security of $\mathsf{3PC}$ simulator.

Corrupt P_1, P_4: $\mathcal{C} = \{P_1, P_4\}$ and $\mathcal{H} = \{P_2, P_3\}$. $\mathcal{S}^{14}_{\mathsf{dm.bobw}}$ playing the role of parties in $\mathcal{H}$ works as follows:

- Act honestly on behalf of P_3 for the commitment instance between P_1 as sender and P_3 as receiver to obtain seed s_2.
- Sample random s_3 and act honestly on behalf of P_2 for the commitment instance between P_2 as sender and P_1 as receiver.
- Sample random s_1 and act honestly on behalf of P_3 and P_2 for the commitment instance between P_3 as sender and P_2 as receiver.

Figure 11.19: Simulator $\mathcal{S}^{14}_{\mathsf{seed.dm}}$

- On behalf of $P_j, j \in \{2,3\}$: Sample $\rho_j \leftarrow \{0,1\}^s$ and act as honest committer in ExtCom with P_{j+1} as receiver.
- For the commitment instance between $P_i, i\{1,4\}$ as sender and P_{i+1} as receiver to commit to string ρ_i:
  - Run the ExtCom protocol with P_i as sender and P_{i+1} as receiver, run rounds 1-3 and broadcast their messages $(\mathsf{extcom}_1^1, \mathsf{extcom}_2^1, \mathsf{extcom}_3^1)$.
  - Rewind the adversary to the end of round 1 for P_4 and P_1 to rerun rounds 2-3 and broadcast $(\mathsf{extcom}_2^2, \mathsf{extcom}_3^2)$.
  - Run extractor algorithm $\mathsf{Extract}$ of the commitment scheme using inputs $(\mathsf{extcom}_1^1, \{\mathsf{extcom}_2^i, \mathsf{extcom}_3^i\}_{i\in[2]})$ to extract the committed string ρ_i.

Figure 11.20: Extraction of challenge-string $\mathcal{S}^{14}_{\mathsf{CCstring.dm}}$

Seed and mask distribution: Run $\mathcal{S}^{14}_{\mathsf{seed.dm}}$ (Fig 11.19) and $\mathcal{S}^{14}_{\mathsf{mask.dm}}$ (Fig 11.16).

Input-sharing: Run $\mathcal{S}^{14}_{\mathsf{ICom.dm}}$ (Fig 11.14)

Cut-and-choose Challenge String: Run $\mathcal{S}^{14}_{\mathsf{CCstring.dm}}$ (Fig 11.20) to compute $\rho = \oplus_{i\in[4]}\rho_i$. Let $\mathsf{CC} = \{k : \rho^k = 1\}$ where ρ^k is used to denote the k^{th} bit of ρ and $\mathsf{EC} = [s] \setminus \mathsf{CC}$.
Run the following three phases for iterator $\mathsf{itr} \in [s]$:

(I) Input-insensitive phase:
- On behalf of honest $P_g, g \in \{2,3\}$ and $h \in \mathcal{S}_g$, do the following steps honestly: Broadcast λ_w^h for output wire w. For every *input* wire w, if w is owned by a garbler $P_i, i \neq g$, broadcast λ_w^i (If P_4 owns w broadcast λ_w^h). For every input wire w, let $k_{w,0}^h$ and $k_{w,1}^h$ denote the two keys derived from seed s_h. For $b \in \{0,1\}$, compute commitments as: $(c[\mathsf{k}]_{w,b}^h, o[\mathsf{k}]_{w,b}^h) \leftarrow \mathsf{Com}(\mathsf{pp}^h, k_{w,b}^h)$ and broadcast $\{c[\mathsf{k}]_{w,b}^h\}$.
- If P_1 broadcasts different copies of λ_w^h (for output wire w) or λ_w^h or $c[\mathsf{k}]_{w,b}^h$ (for some *input* wire w and $b \in \{0,1\}$) for $h \in \mathcal{S}_1$ than what was computed by honest $P_g, g \in \{2,3\}$, then on behalf of P_g: broadcast $o[\mathsf{s}]_h$. Invoke simulator for $\mathsf{3PC}()$ with P_2, P_3, P_4 as participating parties (with P_4 as the corrupt party).
- For input wire w owned by honest $P_i, i \in \{2,3\}$: compute $\lambda_w = (\oplus_{h\in[3]})\lambda_w^h$ using the knowledge of all seeds.

Commitment on masked inputs:
- On behalf of $P_i, i \in \{2,3\}$: For input wire w, with masked input b_w held by parties $P_i, P_j, j \neq i$, compute $(c[b]_w, o[b]_w) = \mathsf{Com}(\mathsf{pp}^{ij}, b_w)$ (using randomness derived from shared ran_{ij}). Broadcast $(\mathsf{pp}^{ij}, c[b]_w)$.
- For input wire w (w.r.t. share x_{ij}) where $i \in \{2,3\}$ and $j \in \{1,4\}$: If P_j broadcasts different

values than computed by P_i, broadcast $o[\mathsf{ran}]_{ji}$, $\{o[\mathsf{s}]_l\}_{l\in\mathcal{S}_i}$ and $o[\mathsf{in}]_{ij}$.

- For input wire w (w.r.t. share held by P_1, P_4, say x_{14}): if P_1, P_4 broadcast mismatching values, receive $o[\mathsf{ran}]_{14}, \{o[\mathsf{s}]_l\}_{l\in\mathcal{S}_j}$ broadcasted by $P_j, j \in \{1,4\}$ and conclude the party (out of the two) by locally computing the messages they broadcasted (say P_1 is corrupt). Invoke simulator for $\mathsf{3PC}()$ accordingly.

Oblivious Transfer: For wire w corresponding to shares possessed by garbler (say P_1) and P_4, say x_{14}:

- Receive b_w sent by P_1 to $\mathcal{F}_{\mathsf{OT}}$ for the OT that is supposed to run between P_1 (as receiver) and P_2 (as sender).
- Compute $x_{14} = b_w \oplus (\oplus_{h\in[3]}\lambda_w^h)$ and $x_1 = x_{12} \oplus x_{13} \oplus x_{14}$. Similarly, compute x_4. Invoke $\mathcal{F}_{\mathsf{uAbort}}$ (Fig 2.3) with $(\mathsf{input}, x_1), (\mathsf{input}, x_4)$ on behalf of P_1^*, P_4^* to obtain y.

Transfer of GC:

- For $\mathsf{itr} \in \mathsf{CC}$, i.e. for GC that will be opened according to the challenge string, behave honestly on behalf of $P_g, g \in \{2,3\}$ to realise $\mathcal{F}_{\mathsf{GCMod}}$ using seeds chosen in seed distribution phase.
- For $\mathsf{itr} \in \mathsf{EC}$, i.e. for GCs that will be evaluated, compute $z = y \oplus \lambda_w$ for the output wire w. Construct a simulated $\mathsf{GC}^1||\mathsf{GC}^2||\mathsf{GC}^3$ using the knowledge of all seeds such that each ciphertext for the output gate of GC^g encrypts the same output key $k_{w,z}^g$.
- Broadcast $\{\mathsf{GC}^h\}$ on behalf of $P_i, i\{2,3\}$ for $h \in \mathcal{S}_i$. If P_1 broadcasts different copies of GC^h than what was computed by honest P_2 (or P_3) for $h \in \mathcal{S}_1$, then on behalf of P_2 (or P_3): broadcast $o[\mathsf{s}]_h$. Invoke simulator for $\mathsf{3PC}()$ with P_2, P_3, P_4 as participating parties (with P_4 as the corrupt party).

(II) Cut-and-choose phase:

- Let the values broadcasted corresponding to seed s_g in the above three phases in iteration itr be denoted by $\mathbf{B}[g]_{\mathsf{itr}}$.
- On behalf of $P_j, j \in \{2,3\}$: receive opening broadcasted by $P_i, i \in \{1,4\}$. If opening sent by P_i is invalid, invoke simulator for $\mathsf{3PC}$ (P_i corrupt). Also, run $\mathsf{ExtOpen}$ routine to broadcast opening for ρ_j.
- For $\mathsf{itr} \in \mathsf{CC}$, the following is done:
  - On behalf of P_2, broadcast openings of seed s_3 and masks ran_{23} and ran_{24}.
  - On behalf of P_3, broadcast openings of seed s_1 and masks ran_{34}.
  - If any opening broadcasted by P_1 is incorrect, run simulator for $\mathsf{3PC}$ with $\mathbb{C}^3 = \mathbf{P} \setminus \{P_1\}$.

(III) Input-sensitive phase: Run the following three phases for $\mathsf{itr} \in \mathsf{EC}$:

Transfer of keys (encoding labels) and masked inputs: For each masked input b_w held by two garblers, do the following on behalf of $P_g, g \in \{2,3\}$:

- If P_g is an owner of w, broadcast $o[b]_w \oplus \mathsf{ran}_{g4}$.
- If P_g is an owner of w, send $o[\mathsf{k}]_{w,b_w}^h, h \in \mathcal{S}_g$ to P_4 on behalf of P_g.

- If P_4 broadcasts $o[\mathsf{ran}]_{i4}$ and (Con, P_i, P_4) w.r.t input wire w, broadcast $o[k]^h_{w,b_w}, h \in \mathcal{S}_g$ on behalf of $P_g, g \neq i$.

Evaluation:

- Receive $\mathbf{Y}$ on behalf of $P_g, g \in [3]$ as broadcasted by P_4. If P_4 does not broadcast anything or if $\mathbf{Y} \neq \{k^h_{w,z}\}_{h\in[3]}$, invoke simulator for $\mathsf{3PC}()$ with P_1, P_2, P_3 as participating parties (with P_1 as the corrupt party).

Figure 11.21: Simulator $\mathcal{S}^{14}_{\mathsf{dm.bobw}}$

Security against active P_1^* and P_4^*: We now argue that $\text{IDEAL}_{\mathcal{F}_{\mathsf{uAbort}},\mathcal{S}^{14}_{\mathsf{dm.bobw}}} \overset{c}{\approx} \text{REAL}_{\pi_{\mathsf{bobw.god}},\mathcal{A}}$ when an adversary $\mathcal{A}$ corrupts two parties P_1 and P_4 (dishonest majority). The views are shown to be indistinguishable via a series of intermediate hybrids.

– HYB_0: Same as $\text{REAL}_{\pi_{\mathsf{bobw.god}},\mathcal{A}}$.

– HYB_1: Same as HYB_0 except: Rerun rounds 2-3 of extractable commitment for the commitment of the challenge string for cut-and-choose (with P_4 as sender and P_1 as receiver) to extract ρ_4. Similarly extract ρ_1. Run the subsequent rounds same as HYB_0.

– HYB_2: Same as HYB_1 except: For share x_{ij} for $i \in \{2,3\}, j \in \{2,3\} \setminus \{i\}$ (i.e. the share that the adversary doesn't get access to), replace $c[\mathsf{in}]_{ij}$ with the commitment of a dummy value.

– HYB_3: Same as HYB_2 except: invoke $\mathcal{F}_{\mathsf{OT}}$ for OT between P_1 as receiver and P_2 as sender to communicate $o[k]^1_{w,b_w}$ for wire w owned by P_1 and P_4.

– HYB_4: Same as HYB_3 except: For $\mathsf{itr} \in \mathsf{EC}$ i.e. for GCs that will be evaluated, construct simulated GC using knowledge of all seeds (instead of constructing an honest GC), in such a way that each ciphertext for the output gate encrypts the same output key which corresponds to $b_w = y \oplus \lambda_w$ where y is obtained after having invoked $\mathcal{F}_{\mathsf{god}}$ and λ_w is known from the information of all seeds.

– HYB_5: Same as HYB_4 except: In HYB_4, $\mathbf{Y}$ is deemed to be invalid if there does not exist a bit b_w such that for each $j \in \mathcal{S}_g$, k^j_w obtained from $\mathbf{Y}$ matches k^j_{w,b_w} while in HYB_5, it $\mathbf{Y}$ is deemed invalid if it is not the one that was encrypted in the simulated GC.

– HYB_6: Same as HYB_5 except: in case of a $P_i, i \in \{1,4\}$ identified to be corrupt, invoke simulator for $\mathsf{3PC}$ instead of running the actual $\mathsf{3PC}$.

Note that $\text{HYB}_5 = \text{IDEAL}_{\mathcal{F}_{\mathsf{god}}, \mathcal{S}^{14}_{\mathsf{dm.bobw}}}$. Next, we show that each pair of hybrids is computationally indistinguishable as follows:

$\text{HYB}_0 \overset{c}{\approx} \text{HYB}_1$: The only difference is that in HYB_1, rounds 2-3 of extractable commitment of the challenge string for cut-and-choose are rerun to extract ρ_4 and ρ_1. Note that the view of the adversary doesn't change across rewinds.

$\text{HYB}_1 \overset{c}{\approx} \text{HYB}_2$: The only difference between the hybrids is that in HYB_2, the commitment for shares x_{ij} for $i \in \{2,3\}, j = \{2,3\} \setminus \{i\}$ are replaced by commitments of dummy values. Note that these are the shares whose openings are not revealed to the adversary. Hence, the indistinguishability follows from the hiding property of the commitment scheme.

$\text{HYB}_2 \overset{c}{\approx} \text{HYB}_3$: Indistinguishability follows from reduction to the underlying OT scheme.
$\text{HYB}_3 \overset{c}{\approx} \text{HYB}_4$: Indistinguishability of hybrids follows from reduction to the security of the underlying garbling scheme which breaks down to the security of PRF. Also, it is ensured that the GCs that are going to be opened for verification have been constructed correctly, so that the adversary cannot distinguish w.r.t. those GCs.

$\text{HYB}_4 \overset{c}{\approx} \text{HYB}_5$: Indistinguishability follows for the two different notions of validity of $\mathbf{Y}$ because a $\mathbf{Y}$ valid according to condition in HYB_5 is valid according to condition in HYB_4. Also a $\mathbf{Y}$ invalid according to condition in HYB_5 can possibly be valid according to condition in HYB_4 only if P_4 could forge the other output keys which is not possible with non-negligible probability according to the authenticity of the garbling scheme.

$\text{HYB}_5 \overset{c}{\approx} \text{HYB}_6$: The indistinguishability follows from the correctness of $\mathsf{3PC}$.

□

Chapter 12

Fairness in Best-of-Both-Worlds Setting

In this chapter, we provide an efficient construction for 4 parties in the BoBW setting that simultaneously guarantees: a) *fairness* in honest majority ($t = 1$), b) *unanimous abort* in dishonest majority ($s = 3$). We discussed that, as per [IKLP06], the best feasible security notions of guaranteed output delivery (against $t < n/2$ corruptions) and unanimous abort (against $s < n$ corruptions) can be attained simultaneously under the additional condition that $t + s < n$. Due to relaxation of security from guaranteed output delivery to fairness, we get rid of this constraint on corruption threshold parameters. Our protocol ideas can be used on top of any 4-party Distributed Garbling scheme that achieves abort security against 3 active corruptions. For concrete instantiation, we use the state-of-the-art n-party scheme of [WRK17] that promises *abort* security against $n-1$ corruptions. We begin with an overview of [WRK17] below.

12.1 Distributed Garbling of [WRK17]

For $n = 4$, [WRK17] has 3 garblers $\{P_1, P_2, P_3\}$ and 1 evaluator P_4 (wlog). Each wire w is associated with a mask $\lambda_w \in \{0, 1\}$ and each party P_i samples its mask share λ_w^i such that $\lambda_w = \oplus_{i\in[4]}\lambda_w^i$. Each garbler P_g also chooses a pair of keys $\mathsf{k}_{w,0}^g, \mathsf{k}_{w,1}^g = \mathsf{k}_{w,0}^g \oplus \Delta^g$ for each wire w where Δ^g is the global offset of P_g. The crux of the construction is the same as the distributed garbling explained in Chapter 10, however they also rely on pair-wise authentication to achieve active security against 3 corruptions. Specifically, [WRK17] uses an optimized, multi-party version of the TinyOT [NNOB12] to generate authentication information (MACs) for every mask share. To elaborate, for its mask-share λ_w^i on wire w, P_i runs a multi-party TinyOT with

every $P_j, j \neq i$ to obtain MAC $\mathsf{M}_j[\lambda_w^i]$ where P_j uses a pair-wise authentication key $\mathsf{K}_j[\lambda_w^i]$ and a global offset Δ^j to authenticate. The keys and mask shares for output wires of XOR gates are set in order to enable free-XOR [KS08]. A garbled-table constructed by P_g for an AND gate with u, v as input wires and w as output wire consists of 4 ciphertexts corresponding to 4 rows (for all possible combinations of the input values on wires u, v). For the ciphertext positioned at row $\mathsf{r}(\mathsf{r} \in [4])$, P_g first obtains authenticated shares of each garbler's keys on w (including her own key share) to be encrypted in row r. P_g then encrypts all these shares collectively with her own keys for input wires u, v corresponding to row r. Note that the authentication mechanism is such that the MAC obtained by P_i w.r.t. the key Along with the keys, the shares of the masked output and their respective MACs are also encrypted in a ciphertext. This construction is formally depicted in functionality $\mathcal{F}_{\mathsf{Pre}}$ (Fig 12.2). During evaluation, P_4, when given the keys for the input wires u, v particular to row r of an AND gate, decrypts the row r of all garbled-tables (wrt each garbler) and computes all keys on output wire w that correspond to the masked output using the shares obtained on decryption. P_4, at every decryption step, verifies if the mask-share and MACs encrypted using her respective authentication key are valid and aborts otherwise. In [WRK17], only the evaluator obtains output using the authenticated mask shares on the output wires (by unmasking the mask on masked output) from each garbler, with MACs ensuring that the received share is valid. We first provide the $\mathcal{F}_{\mathsf{Pre}}$ and $\mathcal{F}_{\mathsf{aBit}}^n$ functionalities used in [WRK17] in Fig 12.2 and 12.1 respectively. Then, we give the formal protocol of [WRK17], $\pi_{\mathsf{dm.abort}}$ for 4 parties in Fig 12.3.

Honest Parties: On receiving $(\mathsf{input}, i, \ell)$ from all parties, pick random string $x \in \{0,1\}^\ell$. For each $j \in [\ell], k \neq i$, pick random $\mathsf{K}_k[x_j]$, compute $\{\mathsf{M}_k[x_j] = \mathsf{K}_k[x_j] \oplus x_j\Delta^k\}_{k\neq i}$. For each $j \in [\ell]$, send $\{\mathsf{M}_k[x_j]\}_{k\neq i}$ to P_i and $\mathsf{K}_k[x_j]$ to P_k for each $k \neq i$.

Corrupted Parties: Corrupted parties can choose their output from the protocol.

Global key queries: The adversary at any point can send some (p, Δ') and will be told if $\Delta' = \Delta^p$.

Figure 12.1: Functionality $\mathcal{F}_{\mathsf{aBit}}^n$

Honest Parties:

- On receiving init from all parties, sample $\{\Delta^i \in \{0,1\}^\kappa\}_{i\in[n]}$ and send Δ^i to P_i.
- On receiving random from all P_i, sample a random bit r and compute authenticated share $\{(r^i, \{\mathsf{M}_j[r^i], \mathsf{K}_i[r^j]\}_{j\neq i})\}_{i\in[n]}$. Send $(r^i, \{\mathsf{M}_j[r^i], \mathsf{K}_i[r^j]\}_{j\neq i})$ to P_i for $i \in [n]$.
- On receiving $(\mathsf{and}, (r^i, \{\mathsf{M}_j[r^i], \mathsf{K}_i[r^j]\}_{j\neq i}), (s^i, \{\mathsf{M}_j[s^i], \mathsf{K}_i[s^j]\}_{j\neq i})$ from P_i for all $i \in [n]$, check whether all MACs are valid. If so, compute $(\oplus_{i\in[n]} r^i)(\oplus_{i\in[n]} s^i)$ and computes authenticated share

$\{(t^i, \{\mathsf{M}_j[t^i], \mathsf{K}_i[t^j]\}_{j\neq i})\}_{i\in[n]}$. Send $(t^i, \{\mathsf{M}_j[t^i], \mathsf{K}_i[t^j]\}_{j\neq i})$ to P_i for $i \in [n]$.

Corrupted Parties: Corrupted parties can choose randomness used to compute the value they receive from the functionality.

Global key queries: The adversary at any point can send some (p, Δ') and will be told if $\Delta' = \Delta^p$.

Figure 12.2: Functionality $\mathcal{F}_{\mathsf{Pre}}$

Inputs and Output: Party P_i has input x_i for $i \in [4]$. P_4 outputs $y = f(x_1, x_2, x_3, x_4)$, each input and output is from $\{0, 1\}$.

Input-independent phase:

- P_i sends init to $\mathcal{F}_{\mathsf{Pre}}$, which sends Δ^i to P_i.
- For each wire w that is either an input wire or output wire of an AND gate, $P_i, i \in [4]$ sends random to $\mathcal{F}_{\mathsf{Pre}}$, which sends $\left(\lambda_w^i, \left\{\mathsf{M}_j[\lambda_w^i], \mathsf{K}_i[\lambda_w^j]\right\}_{j\neq i}\right)$ to P_i s.t $\bigoplus_{i\in[4]} \lambda_w^i = \lambda_w$. $P_i, i \neq 4$ (i.e. just the garblers) also picks a random κ-bit string $\mathsf{k}_{w,0}^i$ as 0-label and sets $\mathsf{k}_{w,1}^i = \mathsf{k}_{w,0}^i \oplus \Delta^i$ as the 1-label.
- For each XOR gate $\mathcal{G}$ with input wires α, β and output wire γ, $P_i, i \in [4]$ computes $\left(\lambda_\gamma^i, \left\{\mathsf{M}_j[\lambda_\gamma^i], \mathsf{K}_i[\lambda_\gamma^j]\right\}_{j\neq i}\right) := \left(\lambda_\alpha^i \oplus \lambda_\beta^i, \left\{\mathsf{M}_j[\lambda_\alpha^i] \oplus \mathsf{M}_j[\lambda_\beta^i], \mathsf{K}_i[\lambda_\alpha^j] \oplus \mathsf{K}_i[\lambda_\beta^j]\right\}_{j\neq i}\right)$. $P_i, i \neq 4$ also computes $\mathsf{k}_{\gamma,0}^i := \mathsf{k}_{\alpha,0} \oplus \mathsf{k}_{\beta,0}$ as 0-label on the output wire. He sets $\mathsf{k}_{\gamma,1}^i = \mathsf{k}_{\gamma,0}^i \oplus \Delta^i$ as 1-label.
- For each AND gate $\mathcal{G}$ with input wires α, β, output wire γ:
  - $P_i, i \in [3]$ sends $\left(\mathsf{and}, \left(\lambda_\alpha^i, \left\{\mathsf{M}_j[\lambda_\alpha^i], \mathsf{K}_i[\lambda_\alpha^j]\right\}_{j\neq i}\right), \left(\lambda_\beta^i, \left\{\mathsf{M}_j[\lambda_\beta^i], \mathsf{K}_i[\lambda_\beta^j]\right\}_{j\neq i}\right)\right)$ to $\mathcal{F}_{\mathsf{Pre}}$ and receives $\left(\lambda_\sigma^i, \left\{\mathsf{M}_j[\lambda_\sigma^i], \mathsf{K}_i[\lambda_\sigma^j]\right\}_{j\neq i}\right)$ for $\bigoplus_{i\in[4]} \lambda_\sigma^i = \left(\bigoplus_{i\in[4]} \lambda_\alpha^i\right)\left(\bigoplus_{i\in[4]} \lambda_\beta^i\right)$.
  - $P_i, i \neq 4$ computes the following locally:
    $\left(\lambda_{\gamma,0}^i, \left\{\mathsf{M}_j[\lambda_{\gamma,0}^i], \mathsf{K}_i[\lambda_{\gamma,0}^j]\right\}_{j\neq i}\right) := \left(\lambda_\sigma^i \oplus \lambda_\gamma^i, \left\{\mathsf{M}_j[\lambda_\sigma^i] \oplus \mathsf{M}_j[\lambda_\gamma^i], \mathsf{K}_i[\lambda_\sigma^j] \oplus \mathsf{K}_i[\lambda_\gamma^j]\right\}_{j\neq i}\right)$
    $\left(\lambda_{\gamma,1}^i, \left\{\mathsf{M}_j[\lambda_{\gamma,1}^i], \mathsf{K}_i[\lambda_{\gamma,1}^j]\right\}_{j\neq i}\right) := \left(\lambda_{\gamma,0}^i \oplus \lambda_\alpha^i, \left\{\mathsf{M}_j[\lambda_{\gamma,0}^i] \oplus \mathsf{M}_j[\lambda_\alpha^i], \mathsf{K}_i[\lambda_{\gamma,0}^j] \oplus \mathsf{K}_i[\lambda_\alpha^j]\right\}_{j\neq i}\right)$
    $\left(\lambda_{\gamma,2}^i, \left\{\mathsf{M}_j[\lambda_{\gamma,2}^i], \mathsf{K}_i[\lambda_{\gamma,2}^j]\right\}_{j\neq i}\right) := \left(\lambda_{\gamma,0}^i \oplus \lambda_\beta^i, \left\{\mathsf{M}_j[\lambda_{\gamma,0}^i] \oplus \mathsf{M}_j[\lambda_\beta^i], \mathsf{K}_i[\lambda_{\gamma,0}^j] \oplus \mathsf{K}_i[\lambda_\beta^j]\right\}_{j\neq i}\right)$
    $\left(\lambda_{\gamma,3}^i, \left\{\mathsf{M}_j[\lambda_{\gamma,3}^i], \mathsf{K}_i[\lambda_{\gamma,3}^j]\right\}_{j\neq i}\right) := \left(\lambda_{\gamma,1}^i \oplus \lambda_\beta^i, \left\{\mathsf{M}_4[\lambda_{\gamma,1}^i] \oplus \mathsf{M}_4[\lambda_\beta^i], \mathsf{K}_i[\lambda_{\gamma,1}^4] \oplus \mathsf{K}_i[\lambda_\beta^4] \oplus \Delta^i\right\} \cup \left\{\mathsf{M}_j[\lambda_{\gamma,1}^i] \oplus \mathsf{M}_j[\lambda_\beta^i], \mathsf{K}_i[\lambda_{\gamma,1}^j] \oplus \mathsf{K}_i[\lambda_\beta^j]\right\}_{j\neq i,4}\right)$
  - P_4 (the evaluator) computes the first three rows as above. The last row is computed as follows:
    $\left(\lambda_{\gamma,3}^4, \left\{\mathsf{M}_j[\lambda_{\gamma,3}^4], \mathsf{K}_4[\lambda_{\gamma,3}^j]\right\}_{j\neq 4}\right) := \left(\lambda_{\gamma,1}^4 \oplus \lambda_\beta^4 \oplus 1, \left\{\mathsf{M}_j[\lambda_{\gamma,1}^4] \oplus \mathsf{M}_j[\lambda_\beta^4], \mathsf{K}_4[\lambda_{\gamma,j}^4] \oplus \mathsf{K}_4[\lambda_\beta^j]\right\}_{j\neq 4}\right)$
  - For each $i \neq 4, P_i$ sends the following to P_4:
    $\mathsf{G}_{\gamma,0}^i := H(\mathsf{k}_{\alpha,0}^i, \mathsf{k}_{\beta,0}^i, \gamma, 0) \oplus \left(\lambda_{\gamma,0}^i, \left\{\mathsf{M}_j[\lambda_{\gamma,0}^i]\right\}_{j\neq i}, \mathsf{k}_{\gamma,0}^i \oplus \left(\bigoplus_{j\neq i} \mathsf{K}_i[\lambda_{\gamma,0}^j]\right) \oplus \lambda_{\gamma,0}^i \Delta^i\right)$
    $\mathsf{G}_{\gamma,1}^i := H(\mathsf{k}_{\alpha,0}^i, \mathsf{k}_{\beta,1}^i, \gamma, 1) \oplus \left(\lambda_{\gamma,1}^i, \left\{\mathsf{M}_j[\lambda_{\gamma,1}^i]\right\}_{j\neq i}, \mathsf{k}_{\gamma,1}^i \oplus \left(\bigoplus_{j\neq i} \mathsf{K}_i[\lambda_{\gamma,1}^j]\right) \oplus \lambda_{\gamma,1}^i \Delta^i\right)$

$\mathsf{G}^i_{\gamma,2} := H(\mathsf{k}^i_{\alpha,1}, \mathsf{k}^i_{\beta,0}, \gamma, 2) \oplus \left(\lambda^i_{\gamma,2}, \left\{\mathsf{M}_j[\lambda^i_{\gamma,2}]\right\}_{j\neq i}, \mathsf{k}^i_{\gamma,2} \oplus \left(\bigoplus_{j\neq i} \mathsf{K}_i[\lambda^j_{\gamma,2}]\right) \oplus \lambda^i_{\gamma,2}\Delta^i\right)$

$\mathsf{G}^i_{\gamma,3} := H(\mathsf{k}^i_{\alpha,1}, \mathsf{k}^i_{\beta,1}, \gamma, 3) \oplus \left(\lambda^i_{\gamma,3}, \left\{\mathsf{M}_j[\lambda^i_{\gamma,3}]\right\}_{j\neq i}, \mathsf{k}^i_{\gamma,3} \oplus \left(\bigoplus_{j\neq i} \mathsf{K}_i[\lambda^j_{\gamma,3}]\right) \oplus \lambda^i_{\gamma,3}\Delta^i\right)$

Input Processing:

- For input wire w that belongs to $P_i, i \neq 4$ (corresponding to input bit $x_w = x_i$), for each $j \neq i$, P_j sends $\left(\lambda^j_w, \mathsf{M}_i[\lambda^j_w]\right)$ to P_i, who checks that $\left(\lambda^j_w, \mathsf{M}_i[\lambda^j_w], \mathsf{K}_i[\lambda^j_w]\right)$ is valid, and computes the masked input value $m_w := x_i \oplus \lambda_w = x^i_w \oplus \left(\bigoplus_{i\in[4]} \lambda^i_w\right)$. P_i broadcasts the masked input value m_w. The garblers i.e. $P_j, j \neq 4$ send k^j_{w,m_w} to P_4.

– For input wire that belongs to P_4 (corresponding to input bit $x_w = x_4$), $P_j, j \neq 4$ sends $\left(\lambda^j_w, \mathsf{M}_1[\lambda^j_w]\right)$ to P_4. P_4 checks if $\left(\lambda^j_w, \mathsf{M}_4[\lambda^j_w], \mathsf{K}_4[\lambda^j_w]\right)$ is valid. If so computes the masked input $m_w := x_4 \oplus \lambda_w = x^4_w \oplus \left(\bigoplus_{i\in[4]} \lambda^i_w\right)$. P_4 sends m_w to P_i who sends k^i_{w,m_w} to P_4.

Circuit Evaluation: P_4 evaluates the circuit in topological order. For each gate $\mathcal{G}$ with input wires α, β and output wire γ, P_4 holds $\left(m_\alpha, \left\{\mathsf{k}^i_{\alpha,m_\alpha}\right\}_{i\neq 4}\right)$ and $\left(m_\beta, \left\{\mathsf{k}^i_{\beta,m_\beta}\right\}_{i\neq 4}\right)$ where $m_\alpha = x_\alpha \oplus \lambda_\alpha$ and $m_\beta = x_\beta \oplus \lambda_\beta$.

– If $\mathcal{G}$ is an XOR gate, P_4 computes $m_\gamma := m_\alpha \oplus m_\beta$ and $\left\{\mathsf{k}^i_{\gamma,m_\gamma} := \mathsf{k}^i_{\alpha,m_\alpha} \oplus \mathsf{k}^i_{\beta,m_\beta}\right\}_{i\neq 4}$

– If $\mathcal{G}$ is an AND gate, P_4 computes $\ell := 2m_\alpha + m_\beta$. For $i \neq 4$, P_4 computes $\left(\lambda^i_{\gamma,\ell}, \left\{\mathsf{M}_j[\lambda^i_{\gamma,\ell}]\right\}_{j\neq i}, \mathsf{k}^i_\gamma\right) := \mathsf{G}^i_{\gamma,\ell} \oplus H\left(\mathsf{k}^i_{\alpha,m_\alpha}, \mathsf{k}^i_{\beta,m_\beta}, \gamma, \ell\right)$.

– P_4 checks that $\left\{\left(\lambda^i_{\gamma,\ell}, \mathsf{M}_4[\lambda^i_{\gamma,\ell}], \mathsf{K}_4[\lambda^i_{\gamma,\ell}]\right)\right\}_{i\neq 4}$ is valid and aborts if fails. P_4 computes $m_\gamma := \bigoplus_{i\in[4]} \lambda^i_{\gamma,\ell}$, and $\left\{\mathsf{k}^i_{\gamma,m_\gamma} := \mathsf{k}^i_\gamma \oplus \left(\bigoplus_{j\neq i} \mathsf{M}_i[\lambda^j_{\gamma,\ell}]\right)\right\}_{i\neq 4}$.

Output processing: For each output wire w: $P_i, i \neq 4$ sends $\left(\lambda^i_w, \mathsf{M}_4[\lambda^i_w]\right)$ to P_4 who checks that $\left(\lambda^i_w, \mathsf{M}_4[\lambda^i_w], \mathsf{K}_4[\lambda^i_w]\right)$ is valid and aborts if fails. P_4 constructs the output mask bit $\lambda_w := \bigoplus_{j\in[4]} \lambda^j_w$ and computes the actual output bit on wire w as $x_w := m_w \oplus \lambda_w$.

Figure 12.3: Distributed Garbling Protocol of [WRK17] $\pi_{\mathsf{dm.abort}}$

12.2 Our Techniques

We now elaborate the challenges and corresponding fixes to transform [WRK17] to provide the desired BoBW guarantees. Firstly, to enable all parties to compute the output, we enforce the evaluator (P_4) to broadcast the masked output shares (obtained on evaluation) along with the MACs (wrt all garblers) in Round 1 of the output phase. A corrupt P_1 can broadcast incorrect MACs wrt some garblers, thereby selectively depriving some garblers of output. To tackle this, we enable each garbler to broadcast an `abort` signal in Round 2, if for some received masked output, its MAC does not pass the authentication check. Although this technique allows the honest parties to be in agreement of the output being computed, the corrupt evaluator still

learns the output as all the mask shares on output wires are available to her violating our primary requirement of fairness when $t = 1$. To handle a possible fairness violation in case of corrupt P_4 who misbehaves in Round 1 of output-computation, we enforce each garbler to release the mask-shares on output wires in Round 3 only if P_1 passes the authentication checks.

The final and major obstruction to fairness in honest majority ($t = 1$) is when a corrupt party sends incorrect (or no) mask-shares in Round 3, but learns the output herself using the mask-shares received from the 3 honest parties. This attack is feasible since the honest parties must rely on the adversary to obtain the output. Hence, we make the honest parties constituting the majority self-sufficient to reconstruct the output mask by additively *re-sharing* (second-level sharing) each authenticated mask-share on all output wires such that 3 parties are enough to reconstruct any output mask-share. Thus, if a faulty party does not release her mask-share, then the remaining 3 (honest) parties can reconstruct the same. In detail, the mask-share λ_w^1 owned by P_1 is split as $\lambda_w^1 = \oplus_{j\in[4]\setminus\{1\}}\lambda_w^{1j}$ where λ_w^{1j} is to be distributed to P_j. Note that even these re-shares need to be authenticated for verified output computation, so we use the $\mathcal{F}_{\mathsf{aBit}}^4$ (Fig 12.1) of [WRK17] to enable P_1 to obtain MACs w.r.t. every P_j for random λ_w^{12} and λ_w^{13}. Using linearity of TinyOT, P_1 computes λ_w^{14} and the corresponding MACs locally while the remaining parties compute the respective authentication keys. P_1 then sends the re-share and MACs (wrt all parties), i.e. $(\lambda_w^{1j}, \{\mathsf{M}_\ell[\lambda_w^{1j}]\}_{\ell\neq 1})$ to P_j who accepts if $(\lambda_w^{1j}, \mathsf{M}_j[\lambda_w^{1j}], \mathsf{K}_j[\lambda_w^{1j}])$ is valid (else aborts).

A non-trivial observation here is that, for fairness guarantee, it is necessary to ensure that all the MACs $\{\mathsf{M}_\ell[\lambda_w^{1j}]\}_{\ell\neq 1}$ received by an honest P_j must be valid to maintain the invariant of self-sufficiency amongst honest parties. Otherwise, a corrupt P_1 may send incorrect $\{\mathsf{M}_\ell[\lambda_w^{1\ell}]\}_{\ell\neq j}$ to P_j which P_j cannot verify (P_j does not own the respective authentication key). Later when P_j sends $(\lambda_w^{1\ell}, \mathsf{M}_\ell[\lambda_w^{1\ell}])$ to P_ℓ in the output phase, P_ℓ (who owns the respective authentication key) rejects it, thus violating fairness. Hence, there is a need to verify the correctness of $\{\mathsf{M}_\ell[\lambda_w^{1j}]\}_{\ell\neq j}$ received by P_j in a secure manner in the re-sharing phase itself to ensure robustness of the output-phase later. We fix this tricky issue as follows: Each party P_i is given an additional authenticated bit $(s_i, \{\mathsf{M}_j[s_i]\}_{j\neq i})$ in the pre-processing phase to use for the MAC verification of re-shares. Now, for verification of (say) $\mathsf{M}_3[\lambda_w^{12}]$ received by P_2 from P_1, P_2 sends $\lambda_w^{12} \oplus s_2$ and $\mathsf{M}_3[\lambda_w^{12}]$ (received from P_1) $\oplus$ $\mathsf{M}_3[s_2]$ (received in pre-processing phase) to P_3 who checks the validity using the key $\mathsf{K}_3[\lambda_w^{12}] \oplus \mathsf{K}_3[s_2]$. Observe that, an incorrect $\mathsf{M}_3[\lambda_w^{12}]$ sent by corrupt P_1 will be detected by P_3 who aborts, causing the protocol to abort before the output phase, thus maintaining fairness. This additional check is performed for all re-shares to ensure that the honest parties receive valid MACs (even corresponding to other parties) of re-shares, thus enabling a robust output reconstruction amongst the honest parties in the face of one corruption.

While the above techniques guarantee *fairness* for honest-majority ($t = 1$), we face a subtle issue in achieving *unanimous abort* for dishonest-majority ($s = 3$). In particular, unanimity is a concern in the case of 2 corruptions, when it is required for the 2 honest parties say, P_α and P_β to be in agreement about the output. Observe that, the above technique of re-sharing each output mask-share such that 3 parties are required to reconstruct the corresponding mask-share, can always enable the adversary (on behalf of 2 corrupt parties) to release incorrect MACs of re-shares selectively to P_α, thus causing P_β to obtain the output and P_α to abort. Similar to the fairness case, this is possible due to the dependency of honest parties on the adversary to provide valid re-shares, thus enabling her to violate unanimity. To get around this subtlety, we employ the tool of Replicated Secret Sharing (RSS) (Chapter 9) of each mask-share (instead of additive) in an authenticated manner such that 2 parties suffice to construct any output mask-share. Specifically, for λ_w^1, P_2 (analogously P_3, P_4) now holds $(\lambda_w^{13}, \{\mathsf{M}_\ell[\lambda_w^{13}]\}_{\ell\neq 1})$ and $(\lambda_w^{14}, \{\mathsf{M}_\ell[\lambda_w^{14}]\}_{\ell\neq 1})$ (all but λ_w^{12}) and their MACs. Each MAC is verified in the re-sharing phase itself using the technique described before. RSS ensures the sufficiency of 2 parties to reconstruct the output mask, hence preserving unanimity in *two* corruption case.

To summarize, the output phase begins with evaluator P_4 broadcasting the masked output shares and the corresponding MACs. This is followed by the garblers broadcasting an `abort` signal if they received an invalid MAC from P_4, in which case, all parties abort. Else, in last round, all parties exchange the authenticated shares, re-shares they own of the output masks to compute the output. The use of broadcast in the output phase allows us to achieve unanimity in case P_4 sends incorrect MACs to a (strict) subset of the honest garblers. The formal protocol, $\pi_{\mathsf{bobw.fair}}$ is presented in Fig 12.4.

Inputs: Party P_i has input x_i for $i \in [4]$.

Output: Each party outputs $y = f(x_1, x_2, x_3, x_4)$ or $\perp$, each input and output is from $\{0, 1\}$.

Common Inputs: The circuit C that takes as input x_i for $i \in [4]$ and computes $f(x_1, x_2, x_3, x_4)$.

Notation: $\mathsf{M}_j[\lambda_w^i]$ denotes the MAC for authentication of bit λ_w^i w.r.t. P_j's key $\mathsf{K}_j[\lambda_w^i]$.

Primitives: 3-party, 1-private RSS, functionalities $\mathcal{F}_{\mathsf{Pre}}, \mathcal{F}_{\mathsf{aBit}}^4$.

Input-independent phase: Run following additional steps on top of Input-Independent phase of $\pi_{\mathsf{dm.abort}}$ (Fig 12.3).

- $P_i, i \in [4]$ sends random to $\mathcal{F}_{\mathsf{Pre}}$ (Fig 12.2) and receives $\left(s_i, \left\{\mathsf{M}_j[s_i], \mathsf{K}_i[s_j]\right\}_{j\neq i}\right)$.
- For output wire w, do the following w.r.t. λ_w^1 (analogously $\lambda_w^2, \lambda_w^3, \lambda_w^4$):
  - Each party invokes $\mathcal{F}_{\mathsf{aBit}}^4$ (Fig 12.1) with $(\mathsf{input}, 1, 2)$ such that P_1 receives $\left(\lambda_w^{12}, \left\{\mathsf{M}_\alpha[\lambda_w^{12}]\right\}_{\alpha\neq 1}\right)$, $\left(\lambda_w^{13}, \left\{\mathsf{M}_\alpha[\lambda_w^{13}]\right\}_{\alpha\neq 1}\right)$ while P_α receives $\mathsf{K}_\alpha[\lambda_w^{12}]$, $\mathsf{K}_\alpha[\lambda_w^{13}]$.

- P_1 computes: $\lambda_w^{14} = \lambda_w^1 \oplus \lambda_w^{12} \oplus \lambda_w^{13}$, $\mathsf{M}_\alpha[\lambda_w^{14}] = \mathsf{M}_\alpha[\lambda_w^1] \oplus \mathsf{M}_\alpha[\lambda_w^{12}] \oplus \mathsf{M}_\alpha[\lambda_w^{13}]$ for $\alpha \neq 1$. P_α computes $\mathsf{K}_\alpha[\lambda_w^{14}] = \mathsf{K}_\alpha[\lambda_w^1] \oplus \mathsf{K}_\alpha[\lambda_w^{12}] \oplus \mathsf{K}_\alpha[\lambda_w^{13}]$.
- For $\alpha \neq 1$ and $\beta \in [4] \setminus \{1, \alpha\}$, P_1 sends $(\lambda_w^{1\beta}, \mathsf{M}_2[\lambda_w^{1\beta}], \mathsf{M}_3[\lambda_w^{1\beta}], \mathsf{M}_4[\lambda_w^{1\beta}])$ to P_α. P_α aborts if $(\lambda_w^{1\beta}, \mathsf{M}_\alpha[\lambda_w^{1\beta}], \mathsf{K}_\alpha[\lambda_w^{1\beta}])$ is invalid for some β.
- P_2 (analogously P_3 and P_4) does the following for the verification of MACs (received from P_1) w.r.t. keys of P_3, P_4: P_2 sends $(\lambda_w^{13} \oplus s_2, \mathsf{M}_\beta[\lambda_w^{13}] \oplus \mathsf{M}_\beta[s_2])$ to P_β for $\beta \in \{3, 4\}$ who aborts if $(\lambda_w^{13} \oplus s_2, \mathsf{M}_\beta[\lambda_w^{13}] \oplus \mathsf{M}_\beta[s_2], \mathsf{K}_\beta[\lambda_w^{13}] \oplus \mathsf{K}_\beta[s_2])$ is invalid. Similar steps are taken for the verification of MACs of λ_w^{14} (received from P_1) w.r.t. keys of P_3, P_4.

Input Processing and **Circuit Evaluation** phase are same as the respective phases of $\pi_{\mathsf{dm.abort}}$.

Output processing: For output wire w, the following is done:

- P_4 (evaluator) obtains $\left(m_w^i, \{\mathsf{M}_j[m_w^i]\}_{j \neq i}\right)_{i \in [3]}$ on evaluation where m_w^i is the masked-output share obtained from GC^i.
- P_4 aborts if $(m_w^i, \mathsf{M}_4[m_w^i], \mathsf{K}_4[m_w^i])$ is invalid, else broadcasts $\left(m_w^i, \{\mathsf{M}_j[m_w^i]\}_{j \neq i}\right)$ for $i \in [4]$.
- $P_g, g \in [3]$ broadcasts `abort`, if $(m_w^i, \mathsf{M}_g[m_w^i], \mathsf{K}_g[m_w^i])$ is invalid for some $i \in [4] \setminus \{g\}$. Each party aborts if an `abort` is broadcast. Else, P_g computes $m_w = \oplus_{j \in [4]} m_w^j$.
- Else P_1 (likewise P_2, P_3 and P_4) sends $\left\{(\lambda_w^1, \mathsf{M}_\alpha[\lambda_w^1]), (\lambda_w^{\beta\alpha}, \mathsf{M}_\alpha[\lambda_w^{\beta\alpha}])\right\}$ privately to P_α for $\alpha \neq 1, \beta = [4] \setminus \{1, \alpha\}$.
- P_1 (analogously P_2, P_3, P_4) computes the output mask bit as:
  - P_1 verifies $(\lambda_w^\alpha, \mathsf{M}_1[\lambda_w^\alpha])$ received from P_α using her key $\mathsf{K}_1[\lambda_w^\alpha]$ for $\alpha \neq 1$. If valid, P_1 accepts the respective share.
  - Else, if the verification of (say) λ_w^2 (received from P_2) did not go through, P_1 checks the validity of $(\lambda_w^{21}, \mathsf{M}_1[\lambda_w^{21}])$ received from $P_\alpha, \alpha \in \{3, 4\}$ using key $\mathsf{K}_1[\lambda_w^{21}]$ and aborts if invalid for each α. Else, P_1 computes $\lambda_w^2 = \lambda_w^{21} \oplus \lambda_w^{23} \oplus \lambda_w^{24}$.
- Each party constructs the output: $y = m_w \oplus (\bigoplus_{j \in [4]} \lambda_w^j)$.

Figure 12.4: Protocol $\pi_{\mathsf{bobw.fair}}$

12.3 Correctness and Security

Theorem 6. *The protocol $\pi_{\mathsf{bobw.fair}}$ securely realizes the functionalities $\mathcal{F}_{\mathsf{fair}}$ (Fig 2.2) and $\mathcal{F}_{\mathsf{uAbort}}$ (Fig 2.3) in the standard model against one active and three active corruptions respectively, assuming enhanced trapdoor permutation.*

12.3.1 Correctness

Lemma 14. *The protocol $\pi_{\mathsf{bobw.fair}}$ is correct.*

Proof. The correctness of the computed output mask and mask-shares on output wires follows from the correctness of [WRK17]. The correctness of re-sharing and authentication follows from

$\mathcal{F}_{\mathsf{Pre}}, \mathcal{F}^4_{\mathsf{aBit}}$. Thus, the correct output y is computed from the masked output and shares/re-shares of output masks. □

12.3.2 Security

Proof. The proof is presented by giving simulators separately for honest majority and dishonest majority case. Let $\mathcal{C}$ be the set of corrupt parties and $\mathcal{H} = [4] \setminus \mathcal{C}$ be the set of honest parties.

Honest Majority: Let $\mathcal{A}$ be a malicious adversary corrupting 1 party in a hybrid model execution of $\pi_{\mathsf{bobw.fair}}$. We discuss the honest majority simulator for two cases: (a) $\mathcal{S}^1_{\mathsf{hm.bobw}}$ when a garbler (say P_1) is corrupt and evaluator P_4 and garblers $\{P_2, P_3\}$ are honest, (b) $\mathcal{S}^4_{\mathsf{hm.bobw}}$ when evaluator P_4 is corrupt and garblers $\{P_1, P_2, P_3\}$ are honest. We now describe the simulator running an ideal-world of the fair functionality $\mathcal{F}_{\mathsf{fair}}$ (Fig 2.2) whose behaviour simulates the behaviour of $\mathcal{A}$.

Malicious P_1: $\mathcal{C} = \{P_1\}$ and $\mathcal{H} = \{P_2, P_3, P_4\}$. $\mathcal{S}^1_{\mathsf{hm.bobw}}$ playing the role of parties in $\mathcal{H}$ works as follows:

$P_1^\star$ is corrupt

Input-Independent phase:

- Act honestly on behalf of parties in $\mathcal{H}$ and play the functionality of $\mathcal{F}_{\mathsf{Pre}}$ (Fig 12.2), recording all outputs (including the re-shares of output masks).
- If an honest party aborts, invoke $\mathcal{F}_{\mathsf{fair}}$ with $(\mathsf{input}, \perp)$ and set $y = \perp$.

Input-Processing phase:

- Run honestly using inputs $\{x_i = 0\}_{P_i \in \mathcal{H}}$.
- If an honest party aborts, invoke $\mathcal{F}_{\mathsf{fair}}$ with $(\mathsf{input}, \perp)$ and set $y = \perp$.
- Receive m_w (where w is the input wire corresponding to P_1's input x_1), and compute $x_1 = m_w \oplus \lambda_w$ (where $\lambda^i_w, i \in [4]$ is obtained on behalf of $\mathcal{F}_{\mathsf{Pre}}$).

Output Computation:

- On behalf of P_4, evaluate the garbled circuit using the keys received to obtain $\left(m^i_w, \{\mathsf{M}_j[m^i_w]\}_{j \neq i}\right)_{i \in [3]}$ where m^i_w is the masked output share obtained from GC^i.
- If $(m^i_w, \mathsf{M}_j[m^i_w], \mathsf{K}_j[m^i_w])$ is invalid for some i and $j : P_j \in \mathcal{H}$, invoke $\mathcal{F}_{\mathsf{fair}}$ with $(\mathsf{input}, \perp)$ and set $y = \perp$. Else, invoke $\mathcal{F}_{\mathsf{fair}}$ with (input, x_1) to obtain output y.
- Compute y' by computing function f on inputs $x_i = 0$ for $i : P_i \in \mathcal{H}$ and x_1 (as extracted above).
- If y and y' are same, send to P_1 actual output mask-shares and the corresponding MACs, else send one share (say corresponding to P_4) and its MAC corresponding to P_1 flipped i.e. instead of $\left(\lambda^4_w, \mathsf{M}_1[\lambda^4_w]\right)$, send $\left(\lambda^4_w \oplus 1, \mathsf{M}_1[\lambda^4_w] \oplus \Delta_1\right)$ and instead of $\left(\lambda^{41}_w, \mathsf{M}_1[\lambda^{41}_w]\right)$, send $\left(\lambda^{41}_w \oplus 1, \mathsf{M}_1[\lambda^{41}_w] \oplus \Delta_1\right)$.

Figure 12.5: Simulator $\mathcal{S}^1_{\mathsf{hm.bobw}}$

Malicious P_4: $\mathcal{C} = \{P_4\}$ and $\mathcal{H} = \{P_1, P_2, P_3\}$. $\mathcal{S}^4_{\mathsf{hm.bobw}}$ playing the role of parties in $\mathcal{H}$ works as follows:

$P_4^\star$ is corrupt

Input-Independent phase:

- Act honestly on behalf of parties in $\mathcal{H}$ and play the functionality of $\mathcal{F}_{\mathsf{Pre}}$ (Fig 12.2), recording all outputs (including the re-shares of output masks).
- If an honest party aborts, invoke $\mathcal{F}_{\mathsf{fair}}$ with $(\mathsf{input}, \perp)$ and set $y = \perp$.

Input-Processing phase:

- Run honestly using inputs $\{x_i = 0\}_{P_i \in \mathcal{H}}$.
- If an honest party aborts, invoke $\mathcal{F}_{\mathsf{fair}}$ with $(\mathsf{input}, \perp)$ and set $y = \perp$.
- Receive m_w (where w is the input wire corresponding to P_4's input x_4), and compute $x_4 = m_w \oplus \lambda_w$ (where $\lambda^i_w, i \in [4]$ is obtained on behalf of $\mathcal{F}_{\mathsf{Pre}}$).

Output Computation:

- Receive $\left(m^i_w, \{\mathsf{M}_j[m^i_w]\}_{j \neq i}\right)_{i \in [4]}$ broadcasted by P_4.
- If $(m^i_w, \mathsf{M}_j[m^i_w], \mathsf{K}_j[m^i_w])$ is invalid for some i and $j : P_j \in \mathcal{H}$, invoke $\mathcal{F}_{\mathsf{fair}}$ with $(\mathsf{input}, \perp)$ and set $y = \perp$. Else, invoke $\mathcal{F}_{\mathsf{fair}}$ with (input, x_4) to obtain output y.
- Compute y' by computing function f on inputs $x_i = 0$ for $i : P_i \in \mathcal{H}$ and x_4 (as extracted above).
- If y and y' are same, send to P_4 actual output mask-shares and the corresponding MACs, else send one share (say corresponding to P_1) and its MAC corresponding to P_4 flipped i.e. instead of $\left(\lambda^1_w, \mathsf{M}_4[\lambda^1_w]\right)$, send $\left(\lambda^1_w \oplus 1, \mathsf{M}_4[\lambda^1_w] \oplus \Delta_4\right)$ and instead of $\left(\lambda^{14}_w, \mathsf{M}_4[\lambda^{14}_w]\right)$, send $\left(\lambda^{14}_w \oplus 1, \mathsf{M}_4[\lambda^{14}_w] \oplus \Delta_4\right)$.

Figure 12.6: Simulator $\mathcal{S}^4_{\mathsf{hm.bobw}}$

Dishonest Majority: Let $\mathcal{A}$ be a malicious adversary corrupting 3 parties in a hybrid model execution of $\pi_{\mathsf{bobw.fair}}$. We discuss the dishonest majority simulator for two cases: (a) $\mathcal{S}^{123}_{\mathsf{dm.bobw}}$ when evaluator P_4 is honest and the three garblers $\{P_1, P_2, P_3\}$ are corrupt, (b) $\mathcal{S}^{234}_{\mathsf{dm.bobw}}$ when a garbler (say P_1) is honest and evaluator P_4 and $\{P_2, P_3\}$ are corrupt. We now describe the simulator running an ideal-world of the unanimous abort functionality $\mathcal{F}_{\mathsf{uAbort}}$ (Fig 2.3) whose behaviour simulates the behaviour of $\mathcal{A}$.

Honest P_4: $\mathcal{C} = \{P_1, P_2, P_3\}$ and $\mathcal{H} = \{P_4\}$. $\mathcal{S}^{123}_{\mathsf{dm.bobw}}$ playing the role of parties in $\mathcal{H}$ (i.e. P_4) works as follows:

$P_1^\star, P_2^\star, P_3^\star$ are corrupt

Input-Independent phase:

- Act honestly on behalf of parties in $\mathcal{H}$ and play the functionality of $\mathcal{F}_{\mathsf{Pre}}$ (Fig 12.2), recording all outputs (including the re-shares of output masks).

- If P_4 would abort, invoke $\mathcal{F}_{\mathsf{uAbort}}$ with $(\mathsf{input}, \perp)$ and set $y = \perp$.

Input-Processing phase:

- Run honestly using input on behalf of P_4 as $\{x_4 = 0\}$.
- If P_4 would abort, invoke $\mathcal{F}_{\mathsf{uAbort}}$ with $(\mathsf{input}, \perp)$ and set $y = \perp$.
- Receive m_w (where w is the input wire corresponding to P_i's input x_i for $P_i \in \mathcal{C}$), and compute $x_i = m_w \oplus \lambda_w$ (where $\lambda_w^j, j \in [4]$ is obtained on behalf of $\mathcal{F}_{\mathsf{Pre}}$).

Output Computation:

- On behalf of P_4, evaluate the garbled circuit using the keys received to obtain $\left(m_w^i, \{\mathsf{M}_j[m_w^i]\}_{j\neq i}\right)_{i\in[3]}$ where m_w^i is the masked output share obtained from GC^i.
- If $(m_w^i, \mathsf{M}_4[m_w^i], \mathsf{K}_4[m_w^i])$ is invalid for some i, invoke $\mathcal{F}_{\mathsf{uAbort}}$ with $(\mathsf{input}, \perp)$ and set $y = \perp$. Else, invoke $\mathcal{F}_{\mathsf{uAbort}}$ with $(\mathsf{input}, x_1), (\mathsf{input}, x_2), (\mathsf{input}, x_3)$ to obtain output y.
- Compute y' by computing function f on inputs $x_4 = 0$ and x_i as extracted above for $i : P_i \in \mathcal{C}$.
- If y and y' are same, broadcast actual masked output shares obtained on evaluation and their corresponding MACs, else, broadcast one share (say corresponding to P_1) and its corresponding MAC flipped i.e. instead of $\left(m_w^1, \{\mathsf{M}_j[m_w^1]\}_{j\neq 1}\right)$, send $\left(m_w^1 \oplus 1, \{\mathsf{M}_j[m_w^1] \oplus \Delta_j\}_{j\neq 1}\right)$.
- If P_4 would abort, invoke $\mathcal{F}_{\mathsf{uAbort}}$ with `abort`, else invoke $\mathcal{F}_{\mathsf{uAbort}}$ with `continue`.

Figure 12.7: Simulator $\mathcal{S}_{\mathsf{dm.bobw}}^{123}$

Honest P_1: $\mathcal{C} = \{P_2, P_3, P_4\}$ and $\mathcal{H} = \{P_1\}$. $\mathcal{S}_{\mathsf{dm.bobw}}^{234}$ playing the role of parties in $\mathcal{H}$ (i.e. P_1) works as follows:

$\underline{P_2^\star, P_3^\star, P_4^\star \text{ are corrupt}}$

Input-Independent phase:

- Act honestly on behalf of parties in $\mathcal{H}$ and play the functionality of $\mathcal{F}_{\mathsf{Pre}}$ (Fig 12.2), recording all outputs (including the re-shares of output masks).
- If P_1 would abort, invoke $\mathcal{F}_{\mathsf{uAbort}}$ with $(\mathsf{input}, \perp)$ and set $y = \perp$.

Input-Processing phase:

- Run honestly using input on behalf of P_1 as $\{x_1 = 0\}$.
- If P_1 would abort, invoke $\mathcal{F}_{\mathsf{uAbort}}$ with $(\mathsf{input}, \perp)$ and set $y = \perp$.
- Receive m_w (where w is the input wire corresponding to P_i's input x_i for $P_i \in \mathcal{C}$), and compute $x_i = m_w \oplus \lambda_w$ (where $\lambda_w^j, j \in [4]$ is obtained on behalf of $\mathcal{F}_{\mathsf{Pre}}$).

Output Computation:

- Invoke FUA with $(\mathsf{input}, x_2), (\mathsf{input}, x_3), (\mathsf{input}, x_4)$ to obtain output y. Receive $\left(m_w^i, \{\mathsf{M}_j[m_w^i]\}_{j\neq i}\right)_{i\in[4]}$ broadcasted by P_4.
- If P_1 would abort, invoke $\mathcal{F}_{\mathsf{uAbort}}$ with `abort`.
- Else, invoke $\mathcal{F}_{\mathsf{uAbort}}$ with `continue`. Compute y' by computing function f on inputs $x_1 = 0$ and

x_i as extracted above for $i : P_i \in \mathcal{C}$.

- If y and y' are same, send to all actual output mask-share and the corresponding MACs, else, send flipped share and the corresponding MAC i.e. instead of $\left(m_w^1, \{\mathsf{M}_j[m_w^1]\}_{j\neq 1}\right)$, send $\left(m_w^1 \oplus 1, \{\mathsf{M}_j[m_w^1] \oplus \Delta_j\}_{j\neq 1}\right)$.

Figure 12.8: Simulator $\mathcal{S}^{234}_{\mathsf{dm.bobw}}$

□

12.4 Scaling to 3 parties

We present scaling of $\pi_{\mathsf{bobw.fair}}$ to 3 parties to provide fairness (for $t = 1$) and unanimous abort (for $s = 2$). However, for 3PC, *unanimous* abort when $s = 2$ is free due the existence of only one honest party. Hence, in 3PC, we strip $\pi_{\mathsf{bobw.fair}}$ of the techniques introduced to handle unanimity concerns in 4PC. Specifically, we replace RSS with the original idea of having *additive* re-shares of each output mask share. We present the formal 3-party protocol in Fig 12.9.

Inputs: Party P_i has input x_i for $i \in [3]$.

Output: Each party outputs $y = f(x_1, x_2, x_3)$ or $\perp$, each input and output is from $\{0, 1\}$.

Common Inputs: The circuit C that takes as input x_i for $i \in [3]$ and computes $f(x_1, x_2, x_3)$.

Notation: $\mathsf{M}_j[\lambda_w^i]$ denotes MAC for λ_w^i w.r.t. P_j's key $\mathsf{K}_j[\lambda_w^i]$.

Primitives: Functionalities $\mathcal{F}_{\mathsf{Pre}}, \mathcal{F}^4_{\mathsf{aBit}}$ (Fig 12.2, 12.1).

Input-independent phase: Run following additional steps on top of Input-Independent phase of $\pi_{\mathsf{dm.abort}}$ (Fig 12.3).

- $P_i, i \in [3]$ sends random to $\mathcal{F}_{\mathsf{Pre}}$ (Fig 12.2) and receives $\left(s_i, \left\{\mathsf{M}_j[s_i], \mathsf{K}_i[s_j]\right\}_{j\neq i}\right)$.
- For each output wire w, do as below w.r.t λ_w^1 (similarly λ_w^2, λ_w^3):
  - Each party invokes $\mathcal{F}^3_{\mathsf{aBit}}$ (Fig 12.1) with $(\mathsf{input}, 1, 1)$ s.t. P_1 receives $\left(\lambda_w^{12}, \left\{\mathsf{M}_\alpha[\lambda_w^{12}]\right\}_{\alpha\neq 1}\right)$ while P_α receives $\mathsf{K}_\alpha[\lambda_w^{12}]$.
  - P_1 sets $\lambda_w^{13} = \lambda_w^1 \oplus \lambda_w^{12}$, $\mathsf{M}_\alpha[\lambda_w^{13}] = \mathsf{M}_\alpha[\lambda_w^1] \oplus \mathsf{M}_\alpha[\lambda_w^{12}]$ for $\alpha \neq 1$. P_α computes $\mathsf{K}_\alpha[\lambda_w^{13}] = \mathsf{K}_\alpha[\lambda_w^1] \oplus \mathsf{K}_\alpha[\lambda_w^{12}]$.
  - P_1 sends $(\lambda_w^{1\alpha}, \mathsf{M}_2[\lambda_w^{1\alpha}], \mathsf{M}_3[\lambda_w^{1\alpha}])$ to $P_\alpha, \alpha \neq 1$. P_α aborts if $\left(\lambda_w^{1\alpha}, \mathsf{M}_\alpha[\lambda_w^{1\alpha}], \mathsf{K}_\alpha[\lambda_w^{1\alpha}]\right)$ is invalid.
  - P_2 (analogously P_3) does the following for the verification of MAC (received from P_1) w.r.t. P_3's key: P_2 sends $\left(\lambda_w^{12} \oplus s_2, \mathsf{M}_3[\lambda_w^{12}] \oplus \mathsf{M}_3[s_2]\right)$ to P_3 who aborts if $\left(\lambda_w^{12} \oplus s_2, \mathsf{M}_3[\lambda_w^{12}] \oplus \mathsf{M}_3[s_2], \mathsf{K}_3[\lambda_w^{12}] \oplus \mathsf{K}_3[s_2]\right)$ is invalid.

Input Processing and **Circuit Evaluation** phase are same as the respective phases of $\pi_{\mathsf{dm.abort}}$.

Output processing: For each output wire w, the following is done:

- P_3 (evaluator) obtains $\left(m_w^i, \{\mathsf{M}_j[m_w^i]\}_{j\neq i}\right)_{i\in[2]}$ on evaluation where m_w^i is the masked-output share obtained from GC^i.
- P_3 aborts if $(m_w^i, \mathsf{M}_3[m_w^i], \mathsf{K}_3[m_w^i])$ is invalid, else broadcasts $\left(m_w^i, \{\mathsf{M}_j[m_w^i]\}_{j\neq i}\right)$ for $i \in [3]$.
- $P_g, g \in [2]$ broadcasts `abort`, if $(m_w^i, \mathsf{M}_g[m_w^i], \mathsf{K}_g[m_w^i])$ is invalid for some $i \in [3] \setminus \{g\}$. All parties abort if an `abort` is broadcast. Else, P_g computes $m_w = \oplus_{j\in[3]} m_w^j$.
- Else P_1 (likewise P_2, P_3) sends $\left\{\left(\lambda_w^1, \mathsf{M}_\alpha[\lambda_w^1]\right), \left(\lambda_w^{\beta 1}, \mathsf{M}_\alpha[\lambda_w^{\beta 1}]\right)\right\}$ privately to P_α for $\alpha \neq 1, \beta = [3] \setminus \{1, \alpha\}$.
- P_1 (analogously P_2, P_3) computes the output mask bit as:
  - P_1 verifies $\left(\lambda_w^\alpha, \mathsf{M}_1[\lambda_w^\alpha]\right)$ received from P_α using her key $\mathsf{K}_1[\lambda_w^\alpha]$ for $\alpha \neq 1$. If valid, P_1 accepts the respective share.
  - Else, if the verification of (say) λ_w^2 (received from P_2) did not go through, P_1 checks the validity of $(\lambda_w^{23}, \mathsf{M}_1[\lambda_w^{23}])$ received from P_3 using key $\mathsf{K}_1[\lambda_w^{23}]$ and aborts if invalid. Else, P_1 computes $\lambda_w^2 = \lambda_w^{21} \oplus \lambda_w^{23}$.
- Each party constructs the output: $y = m_w \oplus (\bigoplus_{j\in[3]} \lambda_w^j)$.

Figure 12.9: 3-party protocol of $\pi_{\mathsf{bobw.fair}}$

Chapter 13

Empirical Results

In this chapter, the detailed implementation results of the BoBW constructions are provided. We begin with the description of hardware and software details of our setup and then elaborate on the performance of our protocols in LAN (local area network) and WAN (wide area network) as our protocols are best suited for high latency networks with constant round complexity. We use AES-128 and SHA-256 as benchmark circuits for the protocol achieving fairness in honest majority, $\pi_{\mathsf{bobw.fair}}$. For $\pi_{\mathsf{bobw.god}}$, we show its application to the widely celebrated system of voting. BoBW protocols are highly relevant for voting application where privacy of individual votes needs to hold irrespective of the number of corruptions, demanding dishonest majority protocols that at best achieve only security with abort. But then, a *single* faulty machine may prevent all honest parties from learning the output by launching a denial-of-service attack; which can be repeatedly carried out over any subsequent re-runs as well. This leads to a requirement of robustness from the protocol, where the adversary cannot cause the execution to abort (despite the adversary herself not learning the outcome), when minority parties are corrupt. Hence, we emphasize the use of robust $\pi_{\mathsf{bobw.god}}$ in voting.

Hardware Details. In LAN setting, our system specifications include 32GB RAM, Intel octa-core processor with processor speed 3.6GHz. The bandwidth in LAN is limited to 1Gbps. In WAN, we use Microsoft Azure D4s_v3 instances located at East Japan, West US, East Australia, South India. The average bandwidth measured is 160Mbps and the slowest link between South India and West US has a round trip time of 0.21 s.

Software Details. The operating system used is Ubuntu 16.04LTS (64-bit). Our code is built in $C{+}{+}11$ standards. We rely on libgarble library built on JustGarble licensed under GNU GPL for garbled circuits and openSSL library for random oracle based instantiations of commitments where Hash is realized using openSSL SHA. Our network emulates a complete

graph and socket connections are used to communicate data between the parties. To implement a robust broadcast channel required in our BoBW protocols, we use Dolev-Strong protocol [DS83] to realize an authenticated broadcast. The public-key signatures needed for this purpose are realized using elliptic-curve based schemes [BDL+12]. We use a multi-threaded environment to improve the performance of our protocols.

Analysis. We highlight our results in terms of computation time (**CT** - time spent computing across all cores), LAN runtime (**LAN** - CT + network time), WAN runtime (**WAN**) and Communication (**COM**). The bracketed values in the tables indicate the overhead needed for worst case run over the honest run. In the BoBW threat model, our protocols are the first attempt in efficiency with small population. We do not compare our BoBW protocols with those in the traditional models of only honest majority or dishonest majority since the security models are incomparable as the notions achieved by our protocols in the BoBW model is best of the security attained in either of the traditional models.

We instantiate the malicious garbling scheme required in our $\pi_{\mathsf{bobw.fair}}$ protocol with the state-of-the-art construction of [WRK17]. Table 13.1 depicts the overhead incurred by $\pi_{\mathsf{bobw.fair}}$ over that of [WRK17] in 4PC and 3PC for AES and SHA circuits. Relying only on the abort security of the underlying garbling scheme, we achieve the strong BoBW guarantees of fairness in 4PC when $t = 1$ and unanimous abort when $s = 3$ with an average minimal overhead of atmost 2.61 `ms`, 2.98 `s` and 3.33 `MB` in LAN, WAN and total COM respectively over [WRK17] that achieves abort for the choice of benchmark circuits. Likewise, in 3PC, the overhead amounts to a mere value of 1.93 `ms`, 2.13 `s` and 0.81 `MB` in **LAN**, **WAN** and **COM** respectively over the choice of benchmark circuits.

Table 13.1: Computation time (**CT**), LAN run-time (**LAN**), WAN run-time (**WAN**) and Communication (**COM**) indicating the additional overhead involved in $\pi_{\mathsf{bobw.fair}}$ protocol over [WRK17] for $g \in [3]$.

Circuit	**#Parties**	**CT(** `ms`**)**		**LAN(** `ms`**)**		**WAN(** `s`**)**		**COM(** `MB`**)**	
		P_g	P_4	P_g	P_4	P_g	P_4	P_g	P_4
AES-128	4PC	0.45	0.64	2.25	2.61	2.95	1.47	0.48	0.28
	3PC	0.34	0.5	1.31	1.83	1.29	1.03	0.12	0.10
SHA-256	4PC	0.68	0.97	2.56	2.5	2.98	1.6	0.94	0.51
	3PC	0.52	0.74	1.93	1.87	2.13	1.31	0.21	0.18

Finally, for benchmarking the performance of $\pi_{\mathsf{bobw.god}}$, we begin with the description of the voting circuit that we use.

Circuit Description. The circuit caters to at most 64 contestants and n voters. The input of each voter can be visualized as a 64-bit input with ith bit corresponding to the i^{th} contestant.

Exactly, one bit is set in the 64-bit input at the position corresponding to the contestant whom the vote is being cast. On receiving the inputs of the voters, the total count of votes for each contestant is calculated and the maximum of the resulting counts is taken to determine the winner. This circuit can also be used for a closed ballot decision making such as passing of a legislative bill with only 2 contestants representing *in favour* and *against*. The decision is chosen to be in favour only if its votes are higher. We first use 64-bit adders and comparators to determine if the inputs of the parties are valid (i.e only one bit is set). We use n-bit adders to compute the count of votes for each contestant and an $\log(n)$-bit Less Than (LT) comparators and 2:1 multiplexer to compare the vote counts for two contestants and select the winner of the pair. We adopt an inverted binary tree technique (merge technique) to pairwise compare the votes and sequentially propagate the winner. For our purpose, we set $n = 4$ (4PC). Table 13.2 depicts the performance and thus practicality of $\pi_{\mathsf{bobw.god}}$ when used for the above described voting circuit.

Table 13.2: Computation time (**CT**), runtime in LAN (**LAN**) and WAN (**WAN**) and Communication (**COM**) of $\pi_{\mathsf{bobw.god}}$ for voting circuit where $P_g, g \in [3]$ denotes a garbler and P_e denotes evaluator.

Circuit	**CT(** ms**)**		**LAN(** ms**)**		**WAN(** s**)**		**COM(** MB**)**	
	P_g	P_e	P_g	P_e	P_g	P_e	P_g	P_e
Honest run	58.91	32.45	62.24	35.87	2.9	2.58	4.76	0.031
Worst Case	+0.84	+0.72	+1.24	+1.15	+1.55	+1.344	+0.086	+0.002

In all the above tables, the difference in the values of AES and SHA circuits is attributed to the bigger circuit, input and output size of SHA compared to AES. For our mixed protocols, although the saving of AES over [CGMV17] is better than SHA, but the factor of saving for SHA is better than AES. Thus, our protocols improve in efficiency for larger circuits.

Chapter 14

Conclusion

In this thesis, we explored the dimensions of the adversary for 4 party computation. We segregate the work into two parts based on the two adversarial models we have considered. The security we discuss are the most desirable notions of fairness and guaranteed output delivery which is possible only in honest majority as per the result of [Cle86]. Hence we extensively cover the threat models where these desirable security notions can be attained.

The first part deals with the standard honest-majority where we closed the gap between the state-of-the-art and the optimal results in terms of communication and computational efficiency. The second part considers a very interesting notion of 'Best of Both Worlds' which explores the security guarantees promised by a protocol simultaneously in honest and dishonest majority. Optimally respecting the feasibility constraints of [IKLP06, Kat07], we provide the first efficient constructions that achieve this supremely desirable security.

The theoretical claims of efficiency in each of our constructions are backed with empirical evidence. For the BoBW construction that provides guaranteed output delivery in honest majority, we show the application to the system of voting which naturally demands a protocol promising BoBW security. Note that the quality of constant rounds makes all protocols in this work suitable for high latency networks such as the Internet.

Bibliography

[ABF+17] Toshinori Araki, Assi Barak, Jun Furukawa, Tamar Lichter, Yehuda Lindell, Ariel Nof, Kazuma Ohara, Adi Watzman, and Or Weinstein. Optimized honest-majority MPC for malicious adversaries - breaking the 1 billion-gate per second barrier. In *2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017*, pages 843–862, 2017. 3

[ACGJ18] Prabhanjan Ananth, Arka Rai Choudhuri, Aarushi Goel, and Abhishek Jain. Round-optimal secure multiparty computation with honest majority. In *CRYPTO*, 2018. 61

[ACJ17] Prabhanjan Ananth, Arka Rai Choudhuri, and Abhishek Jain. A new approach to round-optimal secure multiparty computation. In *CRYPTO*, 2017. 61

[AFL+16] Toshinori Araki, Jun Furukawa, Yehuda Lindell, Ariel Nof, and Kazuma Ohara. High-throughput semi-honest secure three-party computation with an honest majority. In *ACM CCS*, 2016. 3

[AJL+12] Gilad Asharov, Abhishek Jain, Adriana López-Alt, Eran Tromer, Vinod Vaikuntanathan, and Daniel Wichs. Multiparty computation with low communication, computation and interaction via threshold FHE. In *EUROCRYPT*, 2012. 2

[BBC+19] Dan Boneh, Elette Boyle, Henry Corrigan-Gibbs, Niv Gilboa, and Yuval Ishai. How to prove a secret: Zero-knowledge proofs on distributed data via fully linear pcps. *IACR Cryptology ePrint Archive*, 2019. 3

[BCD+09] Peter Bogetoft, Dan Lund Christensen, Ivan Damgård, Martin Geisler, Thomas P. Jakobsen, Mikkel Krøigaard, Janus Dam Nielsen, Jesper Buus Nielsen, Kurt Nielsen, Jakob Pagter, Michael I. Schwartzbach, and Tomas Toft. Secure multiparty computation goes live. In *FC*, 2009. 2

[BDL+12] Daniel J. Bernstein, Niels Duif, Tanja Lange, Peter Schwabe, and Bo-Yin Yang. High-speed high-security signatures. *Journal of Cryptographic Engineering*, 2012. 111

[BDOZ11] Rikke Bendlin, Ivan Damgård, Claudio Orlandi, and Sarah Zakarias. Semi-homomorphic encryption and multiparty computation. In *EUROCRYPT*, 2011. 2

[Bea91] Donald Beaver. Efficient multiparty protocols using circuit randomization. In *CRYPTO*, 1991. 61

[BFO12] Eli Ben-Sasson, Serge Fehr, and Rafail Ostrovsky. Near-linear unconditionally-secure multiparty computation with a dishonest minority. In *CRYPTO*, 2012. 2

[BGJ+18] Saikrishna Badrinarayanan, Vipul Goyal, Abhishek Jain, Yael Tauman Kalai, Dakshita Khurana, and Amit Sahai. Promise zero knowledge and its applications to round optimal MPC. In *CRYPTO*, 2018. 61

[BGW88] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In *STOC*, 1988. 2, 61

[BH07] Zuzana Beerliová-Trubíniová and Martin Hirt. Simple and efficient perfectly-secure asynchronous MPC. In *ASIACRYPT*, 2007. 2

[BH08] Zuzana Beerliová-Trubíniová and Martin Hirt. Perfectly-secure MPC with linear communication complexity. In *TCC*, 2008. 2

[BHP17] Zvika Brakerski, Shai Halevi, and Antigoni Polychroniadou. Four round secure computation without setup. In *TCC*, 2017. 61

[BHR12] Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Foundations of garbled circuits. In *CCS*, 2012. 9, 10

[BIK+17] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H. Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. Practical secure aggregation for privacy-preserving machine learning. In *ACM CCS*, 2017. 62

[BJPR18] Megha Byali, Arun Joseph, Arpita Patra, and Divya Ravi. Fast secure computation for small population over the internet. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, pages 677–694, 2018. xii, 3, 4, 6, 13, 16, 17, 56, 57, 58, 62

[BLO16] Aner Ben-Efraim, Yehuda Lindell, and Eran Omri. Optimizing semi-honest secure multiparty computation for the internet. In *ACM CCS*, pages 578–590, 2016. 68, 71, 72

[BMR90] Donald Beaver, Silvio Micali, and Phillip Rogaway. The round complexity of secure protocols (extended abstract). In *STOC*, 1990. vii, 2, 61, 68

[BTW12] Dan Bogdanov, Riivo Talviste, and Jan Willemson. Deploying secure multi-party computation for financial data analysis - (short paper). In *FC*, 2012. 2, 62

[Can00] Ran Canetti. Security and composition of multiparty cryptographic protocols. *J. Cryptology*, 13(1), 2000. 7

[CCD88] David Chaum, Claude Crépeau, and Ivan Damgård. Multiparty unconditionally secure protocols (extended abstract). In *STOC*, 1988. 61

[CDG87] David Chaum, Ivan Damgård, and Jeroen Graaf. Multiparty computations ensuring privacy of each party's input and correctness of the result. In *CRYPTO*, 1987. 2

[CDI05a] R. Cramer, I. Damgård, and Y. Ishai. Share Conversion, Pseudorandom Secret-Sharing and Applications to Secure Computation. In *TCC*, 2005. 13

[CDI05b] Ronald Cramer, Ivan Damgård, and Yuval Ishai. Share conversion, pseudorandom secret-sharing and applications to secure computation. In Joe Kilian, editor, *Theory of Cryptography*. Springer Berlin Heidelberg, 2005. v, 9

[CGH+18] K. Chida, D. Genkin, K. Hamada, D. Ikarashi, R. Kikuchi, Y. Lindell, and A. Nof. Fast large-scale honest-majority MPC for malicious adversaries. In *CRYPTO*, 2018. 3, 4

[CGMV17] Nishanth Chandran, Juan A. Garay, Payman Mohassel, and Satyanarayana Vusirikala. Efficient, constant-round and actively secure MPC: beyond the three-party case. In *ACM CCS*, 2017. 3, 56, 64, 70, 71, 77, 112

[Cha89] David Chaum. The spymasters double-agent problem: Multiparty computations secure unconditionally from minorities and cryptographically from majorities. In *CRYPTO*, 1989. 63

[CIO98] Giovanni Di Crescenzo, Yuval Ishai, and Rafail Ostrovsky. Non-interactive and non-malleable commitment. In *STOC*, 1998. 12

[CKMZ14] Seung Geol Choi, Jonathan Katz, Alex J. Malozemoff, and Vassilis Zikas. Efficient three-party computation from cut-and-choose. In *CRYPTO*, 2014. 3, 62, 75, 76

[CL14] Ran Cohen and Yehuda Lindell. Fairness versus guaranteed output delivery in secure multiparty computation. In *ASIACRYPT*, 2014. 3, 7, 8

[Cle86] Richard Cleve. Limits on the security of coin flips when half the processors are faulty (extended abstract). In *STOC*, 1986. 2, 61, 113

[DGK09] Ivan Damgård, Martin Geisler, and Mikkel Krøigaard. A correction to 'efficient and secure comparison for on-line auctions'. *IJACT*, 2009. 61

[DN07] Ivan Damgård and Jesper Buus Nielsen. Scalable and unconditionally secure multiparty computation. In *CRYPTO*, 2007. 2, 61

[DO10] Ivan Damgård and Claudio Orlandi. Multiparty computation for dishonest majority: From passive to active security at low cost. In *CRYPTO*, 2010. 2, 61

[DOS18] I. Damgård, C. Orlandi, and M. Simkin. Yet another compiler for active security or: Efficient MPC over arbitrary rings. *CRYPTO*, 2018. 3

[DPSZ12] Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In *CRYPTO*, 2012. 2

[DS83] Danny Dolev and H. Raymond Strong. Authenticated algorithms for byzantine agreement. *SIAM J. Comput.*, 1983. 111

[DSZ15] D. Demmler, T. Schneider, and M. Zohner. ABY - A Framework for Efficient Mixed-Protocol Secure Two-Party Computation. In *NDSS*, 2015. 2

[FHW04] Matthias Fitzi, Thomas Holenstein, and Jürg Wullschleger. Multi-party computation with hybrid security. In *EUROCRYPT*, 2004. 63

[FLNW17] Jun Furukawa, Yehuda Lindell, Ariel Nof, and Or Weinstein. High-throughput secure three-party computation for malicious adversaries and an honest majority. In *EUROCRYPT*, 2017. 3

[GGHR14] Sanjam Garg, Craig Gentry, Shai Halevi, and Mariana Raykova. Two-round secure MPC from indistinguishability obfuscation. In *TCC*, 2014. 2, 61

[GGR18] Daniel Genkin, S. Dov Gordon, and Samuel Ranellucci. Best of both worlds in secure computation, with low communication overhead. In *ACNS*, 2018. 64

[GLNP15] Shay Gueron, Yehuda Lindell, Ariel Nof, and Benny Pinkas. Fast garbling of circuits under standard assumptions. In *ACM CCS*, 2015. 9

[GLS15] S. Dov Gordon, Feng-Hao Liu, and Elaine Shi. Constant-round MPC with fairness and guarantee of output delivery. In *CRYPTO*, 2015. 8

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In *STOC*, 1987. 2, 61

[Gol01] Oded Goldreich. *The Foundations of Cryptography - Volume 1, Basic Techniques*. Cambridge University Press, 2001. 7

[GRW18] S. Dov Gordon, Samuel Ranellucci, and Xiao Wang. Secure computation with low communication from cross-checking. *IACR Cryptology ePrint Archive*, 2018:216, 2018. 3

[HHPV18] Shai Halevi, Carmit Hazay, Antigoni Polychroniadou, and Muthuramakrishnan Venkitasubramaniam. Round-optimal secure multi-party computation. In *CRYPTO*, 2018. 61

[HIKR18] Shai Halevi, Yuval Ishai, Eyal Kushilevitz, and Tal Rabin. Best possible information-theoretic MPC. In *TCC*, 2018. 63

[HLM13] Martin Hirt, Christoph Lucas, and Ueli Maurer. A dynamic tradeoff between active and passive corruptions in secure multi-party computation. In *CRYPTO*, 2013. 63

[HLMR11] Martin Hirt, Christoph Lucas, Ueli Maurer, and Dominik Raub. Graceful degradation in multi-party computation (extended abstract). In *ICITS*, 2011. 63

[HLMR12] Martin Hirt, Christoph Lucas, Ueli Maurer, and Dominik Raub. Passive corruption in statistical multi-party computation - (extended abstract). In *ICITS*, 2012. 63

[HMZ08] Martin Hirt, Ueli M. Maurer, and Vassilis Zikas. MPC vs. SFE : Unconditional and computational security. In *ASIACRYPT*, 2008. 63

[IKK+11] Yuval Ishai, Jonathan Katz, Eyal Kushilevitz, Yehuda Lindell, and Erez Petrank. On achieving the "best of both worlds" in secure multiparty computation. *SIAM J. Comput.*, 2011. 61, 62, 63

[IKKP15] Yuval Ishai, Ranjit Kumaresan, Eyal Kushilevitz, and Anat Paskin-Cherniavsky. Secure computation with minimal interaction, revisited. In *CRYPTO*, 2015. 3, 4, 5, 13, 39, 41, 56, 57, 58

[IKLP06] Yuval Ishai, Eyal Kushilevitz, Yehuda Lindell, and Erez Petrank. On combining privacy with guaranteed output delivery in secure multiparty computation. In *CRYPTO*, 2006. 61, 63, 64, 73, 98, 113

[ISN89] Mitsuru Ito, Akira Saito, and Takao Nishizeki. Secret sharing scheme realizing general access structure. *Electronics and Communications in Japan (Part III: Fundamental Electronic Science)*, 1989. v, 9

[JKO13] Marek Jawurek, Florian Kerschbaum, and Claudio Orlandi. Zero-knowledge using garbled circuits: how to prove non-algebraic statements efficiently. In *CCS*, 2013. 9

[Kat07] Jonathan Katz. On achieving the "best of both worlds" in secure multiparty computation. In *ACM Symposium on Theory of Computing*, 2007. 61, 62, 63, 113

[KMO01] Jonathan Katz, Steven Myers, and Rafail Ostrovsky. Cryptographic counters and applications to electronic voting. In *EUROCRYPT*, 2001. 61

[KS08] Vladimir Kolesnikov and Thomas Schneider. Improved garbled circuit: Free XOR gates and applications. In *ICALP*, 2008. 68, 99

[Lin13] Yehuda Lindell. Fast cut-and-choose based protocols for malicious and covert adversaries. In *CRYPTO*, 2013. 75

[Lin16] Yehuda Lindell. Fast cut-and-choose-based protocols for malicious and covert adversaries. *J. Cryptology*, 2016. 64, 76

[Lin17] Yehuda Lindell. How to simulate it - A tutorial on the simulation proof technique. In *Tutorials on the Foundations of Cryptography.*, pages 277–346. 2017. 7

[LP07] Yehuda Lindell and Benny Pinkas. An efficient protocol for secure two-party computation in the presence of malicious adversaries. In *EUROCRYPT*, 2007. 78

[LP11] Yehuda Lindell and Benny Pinkas. Secure two-party computation via cut-and-choose oblivious transfer. In *TCC*, 2011. 76

[LPSY15] Yehuda Lindell, Benny Pinkas, Nigel P. Smart, and Avishay Yanai. Efficient constant round multi-party computation combining BMR and SPDZ. In *CRYPTO*, 2015. 2

[LRM10] Christoph Lucas, Dominik Raub, and Ueli M. Maurer. Hybrid-secure MPC: trading information-theoretic robustness for computational privacy. In *PODC*, 2010. 62, 63, 64

[MF06] Payman Mohassel and Matthew K. Franklin. Efficiency tradeoffs for malicious two-party computation. In *PKC*, 2006. 76

[MR18] P. Mohassel and P. Rindal. ABY^3: A Mixed Protocol Framework for Machine Learning. In *ACM CCS*, 2018. 2, 62

[MRZ15] Payman Mohassel, Mike Rosulek, and Ye Zhang. Fast and secure three-party computation: The garbled circuit approach. In *ACM CCS*, 2015. 2, 3, 4, 6, 16, 17, 56, 57, 58, 64, 77, 79

[MZ17] Payman Mohassel and Yupeng Zhang. Secureml: A system for scalable privacy-preserving machine learning. In *IEEE Symposium on Security and Privacy*, 2017. 62

[Nao91] Moni Naor. Bit commitment using pseudorandomness. *J. Cryptology*, 4(2), 1991. 12

[NBK15] Divya G. Nair, V. P. Binu, and G. Santhosh Kumar. An improved e-voting scheme using secret sharing based secure multi-party computation. *CoRR*, 2015. 61

[NNOB12] Jesper Buus Nielsen, Peter Sebastian Nordholt, Claudio Orlandi, and Sai Sheshank Burra. A new approach to practical active-secure two-party computation. In *CRYPTO*, 2012. 64, 98

[NV18] P. S. Nordholt and M. Veeningen. Minimising Communication in Honest-Majority MPC by Batchwise Multiplication Verification. In *ACNS*, 2018. 4

[PR18] Arpita Patra and Divya Ravi. On the exact round complexity of secure three-party computation. Cryptology ePrint Archive, Report 2018/481, 2018. https://eprint.iacr.org/2018/481. 3, 62

[PRS02] Manoj Prabhakaran, Alon Rosen, and Amit Sahai. Concurrent zero knowledge with logarithmic round-complexity. In *(FOCS*, 2002. 66

[PW09] Rafael Pass and Hoeteck Wee. Black-box constructions of two-party protocols from one-way functions. In *TCC*, 2009. 66

[RB89] Tal Rabin and Michael Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority (extended abstract). In *STOC*, 1989. 2, 61

[Ros04] Alon Rosen. A note on constant-round zero-knowledge proofs for NP. In *TCC*, 2004. 66

[RS04] Phillip Rogaway and Thomas Shrimpton. Cryptographic hash-function basics: Definitions, implications, and separations for preimage resistance, second-preimage resistance, and collision resistance. In *FSE*, 2004. v, 9

[WRK17] Xiao Wang, Samuel Ranellucci, and Jonathan Katz. Global-scale secure multiparty computation. Cryptology ePrint Archive, Report 2017/189, 2017. https://eprint.iacr.org/2017/189. vii, x, xii, 65, 70, 79, 98, 99, 101, 102, 104, 111

[Yao82] Andrew Chi-Chih Yao. Protocols for secure computations (extended abstract). In *FOCS*, 1982. 2, 16

[ZRE15] Samee Zahur, Mike Rosulek, and David Evans. Two halves make a whole - reducing data transfer in garbled circuits using half gates. In *EUROCRYPT*, 2015. 9, 56